Etude de protocoles de diffusion atomique

Emmanuelle Anceaume, Pascale Minet

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Rapports de Recherche

1992

25ème anniversaire

N° 1774

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

ETUDE DE PROTOCOLES DE DIFFUSION ATOMIQUE

Emmanuelle ANCEAUME
Pascale MINET

Octobre 1992

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

RR-1774

Rapports de Recherche

Programme 1
Réseaux et Systèmes Répartis

ETUDE DE PROTOCOLES DE DIFFUSION ATOMIQUE

A STUDY OF ATOMIC BROADCAST PROTOCOLS

Emmanuelle Anceaume
Pascale Minet

Octobre 1992

ETUDE DE PROTOCOLES DE DIFFUSION ATOMIQUE

Emmanuelle Anceaume - Pascale Minet

Projet Reflecs, INRIA,
Rocquencourt,
BP 105, 78153 Le Chesnay Cedex

RESUME. Dans cet article, nous étudions les protocoles de diffusion atomique les plus représentatifs de l'état de l'art. Huit protocoles de diffusion atomique [BJ87], [BSS90], [CAS85], [CM84], [GL90], [GT89], [KTH89] et [VRB89] sont analysés. Après avoir précisé les motivations de cette étude, nous définissons la problématique de la diffusion atomique. Les protocoles étudiés sont décrits selon une grille d'évaluation qui met l'accent sur cinq propriétés que nous avons définies, et qui nous apparaissent nécessaires dans un système réparti tolérant aux fautes. Les protocoles de diffusion atomique sont classés suivant le type de contrôle (centralisé ou décentralisé), et le mode de défaillance toléré. Les caractéristiques de chaque protocole de diffusion font l'objet d'un tableau récapitulatif.

MOTS CLES. Diffusion atomique, consensus unanime, ordre total,terminaison, diffusion spontanée, diffusions concurrentes, exclusion abusive, non contamination, contrôle centralisé.

A STUDY OF ATOMIC BROADCAST PROTOCOLS

ABSTRACT. Eight of the most significant atomic broadcast protocols [BJ87], [BSS90], [CAS85], [CM84], [GL90], [GT89], [KTH89] and [VRB89] are analysed. Motivations for the study, and the essence of the atomic broadcast problem are presented. The protocols are described in terms of a common framework. Five properties (spontaneous broadcast, concurrent broadcasts, minimum number of aborts, no wrong exclusion and no contamination), which appear to be necessary in a reliable distributed system are described. Protocols are distinguished according to the type of control used (centralized or decentralized), and according to their tolerated failure modes. The principal features of each atomic broadcast protocol are summarized in a table.

KEYWORDS. Atomic broadcast, unanimous agreement, total order, termination, spontaneous broadcast, concurrent broadcasts, wrong exclusion, no contamination, centralized control, decentralized control.

1. Généralités

Les protocoles de diffusion atomique constituent un composant essentiel des sytèmes tolérants aux fautes. Par définition, ces protocoles doivent garantir que tous les processeurs non fautifs remettent la même séquence ordonnée de messages. Une définition plus formelle de la diffusion atomique est donnée dans le paragraphe 2.2. Les protocoles de diffusion atomique sont au coeur de problèmes tels que la validation fiable, les opérations atomiques [GRA78], l'appartenance à un groupe [CDD90], [RB91] etc. Le rôle des protocoles de diffusion atomique dans les systèmes tolérants aux fautes a été comparé à celui de la communication par messages dans les systèmes d'exploitation [BJ87].

2. Problématique

2.1. Fourniture du service de diffusion

Nous nous intéressons dans ce paragraphe à la fourniture du service de diffusion atomique. L'utilisateur du service de diffusion est appelé **hôte** (ou encore processeur d'application). Le service de diffusion atomique au sein d'un groupe de diffusion est fourni par les **processeurs de communication** membres de ce groupe.

Relativement à une invocation du service de demande de diffusion, un processeur membre du groupe joue soit le rôle de l'émetteur, soit le rôle d'un récepteur. Seuls du membres du groupe de diffusion sont autorisés à être émetteurs.

Le service de diffusion est illustré sur la figure 1. Il utilise les conventions de service du modèle de référence OSI de l'ISO.

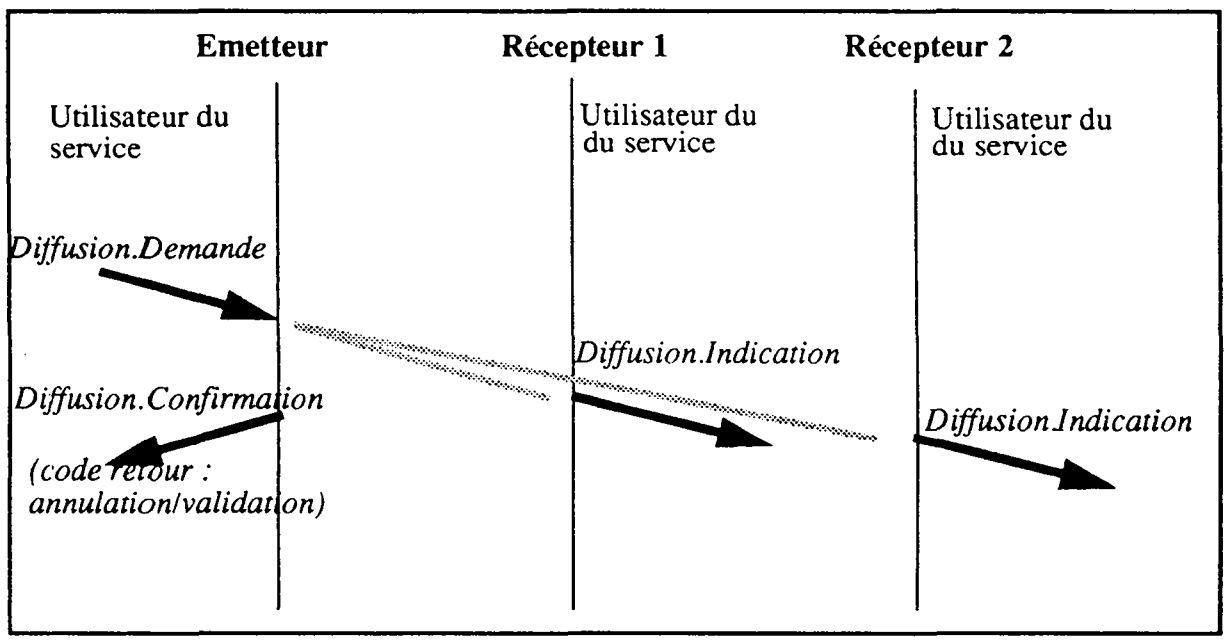

figure 1. Description du service Diffusion

Notons cependant, que dans la littérature, la primitive Diffusion.Demande est dénommée **Demande de diffusion**, et la primitive Diffusion.Indication est dénommée **remise** ou **délivrance d'un message.**

Considérons maintenant l'entité chargée d'implémenter le protocole de diffusion atomique. Cette entité est appelée **entité de diffusion**. Ses interactions sont représentées sur la figure 2.

A titre d'exemple, nous décrivons le comportement de cette entité dans le cas d'un protocole de diffusion atomique en deux phases (cf paragraphe 4.).

Dans la première phase :

- l'hôte émetteur adresse une demande de diffusion (primitive *Demande de diffusion*) à son entité de diffusion, laquelle utilise pour diffuser le message (primitive *diffuse*) soit les services de communication du niveau MAC (Medium Access Control) ([CM84], [CAS85], [PBS89], [VRB89] et [MA91]), soit les services de communication du niveau Transport ([BJ87]).

- l'entité de diffusion d'un récepteur reçoit ce message (primitive *reçoit*), le traite et émet un acquittement (primitive *émet*). Cette primitive utilise les services fournis par son entité de communication.

- l'entité de diffusion émetteur reçoit cet acquittement (primitive *reçoit*). Lorsque cette entité de diffusion émetteur a reçu tous les acquittements des entités de diffusion récepteur, elle entre en deuxième phase du protocole.

Dans la deuxième phase :

- l'entité de diffusion émetteur diffuse (primitive *diffuse*) la validation, et invoque la primitive de *Confirmation*.

- l'entité de diffusion d'un récepteur (primitive *reçoit*) reçoit cette validation et remet (primitive *Remise*) le message à son hôte.

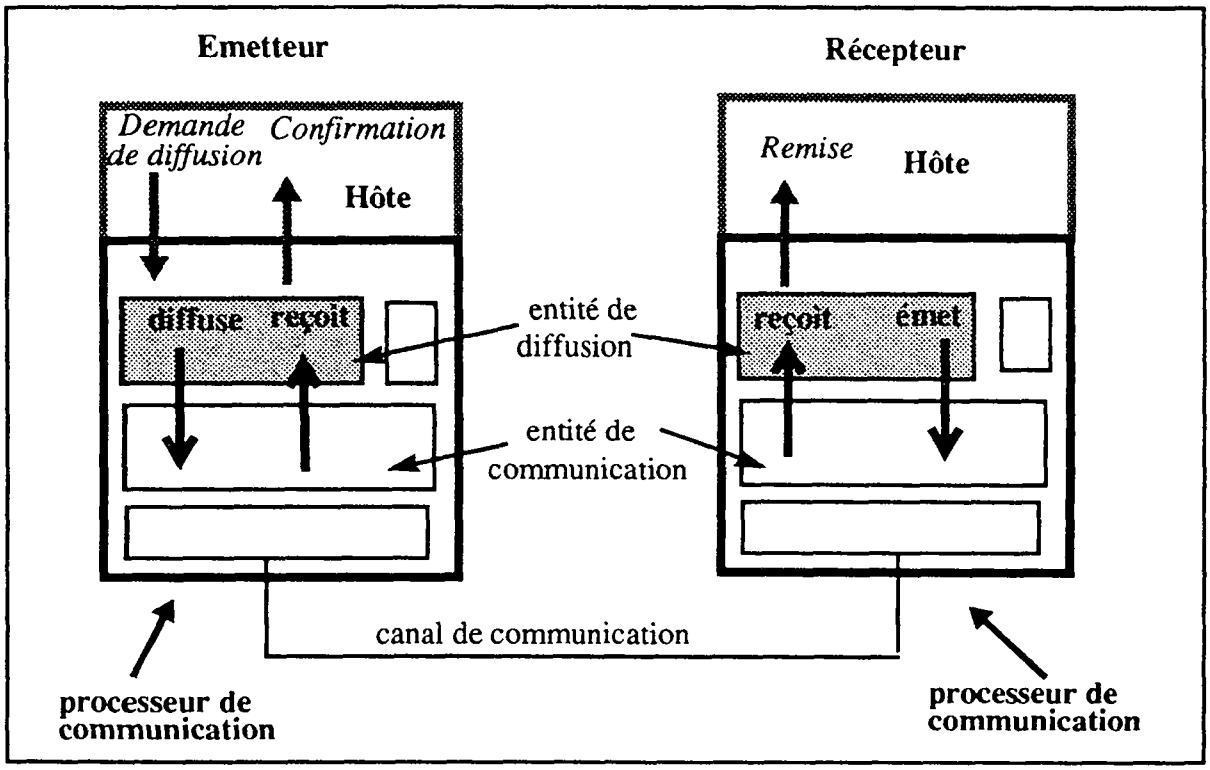

figure 2. Interactions de l'entité de diffusion

Dans la suite de ce document, certaines facilités de notation sont adoptées :

- les termes "remettre" et "remise" sont synonymes.
- un processeur de communication est appelé processeur et est noté ***Pi***,
- émetteur ou processeur émetteur sont synonymes,
- récepteur ou processeur récepteur sont synonymes.

2.2. Définition d'un protocole de diffusion atomique

La définition formelle d'un protocole de diffusion atomique est basée sur les propriétés caractéristiques suivantes (notons que la signification des termes "processeur correct" utilisés ci-dessous, dépend du mode de défaillance supposé par le protocole de diffusion, cf paragraphe 3.2.) :

- **Consensus unanime** : si un processeur correct diffuse un message alors soit tous les processeurs corrects remettent le message à leur processeur d'application, soit aucun d'entre eux ne le remet. Cette propriété est définie dans [CAS85],
- **Terminaison** : un protocole de diffusion possède la propriété de terminaison si chaque processeur correct connaît l'issue de la diffusion en un temps fini. Cette propriété est définie dans [CAS85].
- **Ordre total** en présence d'un seul groupe de diffusion : tous les processeurs corrects du groupe remettent les messages dans le même ordre.

Par définition, un **protocole de diffusion atomique** ([CAS85], [VRB89], [BJ87], [LG90], et [MA91]) garantit les trois propriétés caractéristiques de Consensus unanime, Terminaison et Ordre total. Certains protocoles de diffusion atomique offrent en plus le Consensus Uniforme (voir remarque 1).

Remarque 1 :

Une importante remarque concernant le consensus unanime peut être faite. Le consensus unanime ne concerne que les processeurs corrects, rien n'est dit sur le comportement des processeurs fautifs. Considérons par exemple un système transactionnel réparti où il est impératif que la séquence des transactions validées sur un site qui a défailli soit cohérente avec la séquence des transactions validées par les sites corrects. Plus précisément afin de garantir l'atomicité des transactions, une transaction ne doit pas apparaitre comme validée par un site qui a par la suite défailli alors qu'elle a été annulée par les sites corrects, et inversement. Ceci est donc l'objet de la propriété de consensus uniforme :

- **Consensus uniforme** : si un processeur correct ou incorrect remet un message à son processeur d'application, alors tous les processeurs corrects remettent ce message à leur processeur d'application. Cette propriété est définie dans [GT89] et [HAD90].

Remarque 2 :

La propriété d'ordre énoncée précédemment était relative à un seul groupe de diffusion. Dans le cas où plusieurs groupes de diffusions coexistent dans le système et qu'un processeur appartient à plusieurs de ces groupes de diffusion, il est nécessaire de considérer l'ordre des messages diffusés dans ces groupes. Quel est l'ordre garanti chez les processeurs appartenant à l'intersection de deux groupes de diffusion ? Plus précisément, considérons deux groupes de diffusions *g1* et *g2* quelconques, alors le protocole de diffusion peut offrir trois types d'ordre :

• soit aucun ordre n'est garanti au sein de $g1 \cap g2$: deux processeurs corrects appartenant à $g1 \cap g2$ peuvent délivrer un message diffusé dans *g1* et un message diffusé dans *g2* dans un ordre relatif quelconque (cf figure 3.),

• soit l'ordre total est garanti au sein de $g1 \cap g2$: tous les processeurs corrects appartenant à $g1 \cap g2$ délivrent les messages diffusés dans *g1* ou *g2* dans le même ordre,

• soit l'ordre causal est garanti au sein de $g1 \cap g2$: tous les processeurs appartenant à $g1 \cap g2$ délivrent les messages causalement liés diffusés dans *g1* ou *g2* dans le même ordre.

Dans la figure ci-dessous, les processeurs *P1, P2* et *P3* appartiennent au groupe *g1*, et les processeurs *P2, P3* et *P4* appartiennent au groupe *g2*. Les messages *m1* et *m2* sont diffusés dans le groupe *g1* et les messages *m3* et *m4* sont diffusés dans le groupe g2. Le processeur *P2* remet les messages dans l'ordre : *m1<m2<m3<m4*, tandis que le processeur *P3* remet les messages dans l'ordre : *m1<m3<m4<m2*.

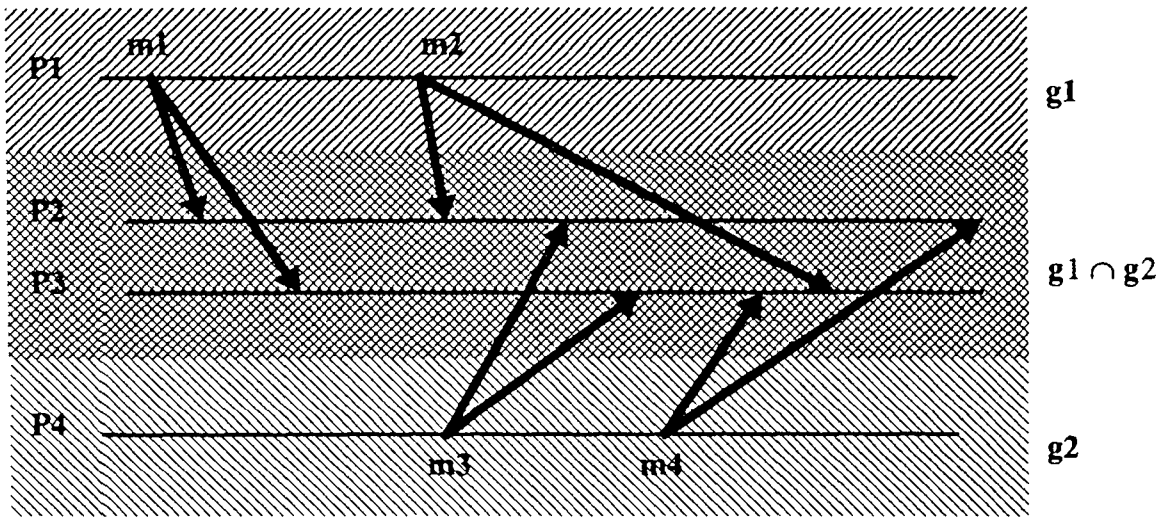

figure 3. Ordre garanti au sein de l'intersection de deux groupes de diffusion

Dans la suite, nous nous sommes limités au cas d'un seul groupe de diffusion.

2.3. Modèle à délai borné et modèle à délai non borné

Deux modèles de systèmes répartis peuvent être considérés : soit un **modèle à délai non borné**, soit un **modèle à délai borné**. Dans le premier cas, la caractéristique essentielle de ces systèmes est l'incapacité que l'on a de donner une borne à priori d'une part sur les temps de transmission des messages et d'autre part sur les délais de traitement. La terminologie employée dans la littérature désigne ce type de système comme un système **asynchrone**. Dans le cas où l'on connait à priori, une telle borne on parle de systèmes synchrones. Comme on le voit, cette définition fait appel explicitement à la connaissance d'un temps physique avec lequel on peut définir une borne supérieure sur les temps de transmission d'un message et sur les temps de traitement. Il existe plusieurs manières d'obtenir ce temps physique. L'une d'entre elles, consiste à faire l'hypothèse que l'évolution du réseau est réglée par une horloge globale qui produit des pulsations globales [GT89] ; tous les temps sont alors mesurés à l'aide de cette

horloge globale, supposée ultra-fiable. Une autre manière utilisée par Cristian est basée sur la synchronisation d'horloges locales ; la dérive maximale entre deux horloges est maintenue inférieure à une borne ε par un algorithme de synchronisation d'horloges. Un message émis sur un canal à la pulsation locale t est reçu à l'autre extrémité du canal au plus tard à la pulsation locale $t+u$, (mesurée chez le destinataire). Le nombre maximal u de pulsations écoulées entre l'envoi et la réception est borné a priori et cette borne (prenant en compte ε) est connue initialement de tous les sites [CRI89]. Une troisième manière consiste à utiliser des horloges locales non-synchronisées pour mesurer localement le délai d'un aller-retour et ignorer les réponses parvenues après un délai supérieur à deux fois la borne supérieure de transmission.

2.3.1. Possible ou impossible ?

Le problème de la diffusion atomique n'a pas de solution dans un système à délai non borné : on ne peut garantir la propriété de terminaison en un temps fini, ne pouvant distinguer le défaut de transmission d'un message, d'un temps de transmission infiniment long. Faisant suite à cette observation, il a été montré en 1985 par Fischer, Lynch et Paterson [FLP85] qu'il est impossible d'obtenir un consensus réparti fiable dans un modèle à "délai non borné", et ceci même en présence d'un seul processeur fautif. Remarquons que ce résultat d'impossibilité a été établi en fait pour un modèle très général (inexistence de borne) et que pour casser ce résultat d'impossibilité, il est intéressant d'étudier la faisabilité du problème en introduisant un minimum d'hypothèses supplémentaires.

2.4. Protocoles déterministes ou probabilistes

Un protocole ou algorithme est dit déterministe au sens où il n'utilise jamais de tirage de nombre aléatoire pour définir son comportement : celui-ci ne dépend que de son état initial et des entrées reçues. Une façon d'essayer de donner des solutions aux problèmes qui n'ont pas de solution déterministe est de considérer qu'un processeur peut déterminer son comportement en tirant des nombres aléatoires, on parle alors d'algorithmes probabilistes. Dans ce cas la propriété de terminaison qui informellement était : "la terminaison est atteinte en x (valeur bornée) phases" devient "la probabilité qu'un processeur n'ait pas terminé en y phases tend vers 0 lorsque y tend vers l'infini".

Dans la suite, nous ne nous intéressons qu'aux protocoles déterministes conçus et mis en oeuvre pour un modèle à délai borné.

3. Grille d'évaluation des protocoles de diffusion

Tous les protocoles de diffusion que nous allons présenter par la suite, sont décrits selon une grille d'évaluation commune. Nous distinguons :

- **les hypothèses** : architecture du système, mode de défaillance ...
- **le fonctionnement du protocole** : en l'absence de défaillance, sur défaillance de l'émetteur, d'un récepteur ...
- **et le comportement d'un protocole de diffusion atomique vis-à-vis de cinq propriétés** essentielles pour un système réparti tolérant aux fautes.

3.1. Architecture du système

Les protocoles de diffusion font généralement référence à l'architecture suivante. Les processeurs qui implémentent le protocole de diffusion sont interconnectés par un médium de communication (canal point-à-point ou canal à diffusion), ne partagent pas de mémoire commune et communiquent par envoi de messages. Chaque processeur de communication est associé à un processeur d'application (encore appelé hôte), et a accès à une horloge (locale ou globale).

Plus précisément, les processeurs de communication sont supposés accéder :

- soit à **une horloge globale unique** : elle est supposée ultra-fiable [GT89],
- soit à **des horloges locales synchronisées** : chaque processeur dispose alors de sa propre horloge locale. Toutes les horloges locales sont synchronisées par l'intermédiaire d'un protocole de synchronisation d'horloges.
- soit à **des horloges locales non synchronisées** : chaque processeur dispose de sa propre horloge locale et la dérive maximale entre deux horloges est bornée pendant la durée d'une phase du protocole de diffusion.

Les délais de transmission et de traitement sont supposés bornés (le modèle temps non borné n'a pas été abordé au cours de cette étude).

Le protocole de diffusion utilise des **services de communication** : soit MAC (Medium Access Control) ([CM84], [CAS85], [PBS89], [VRB89] et [MA91]), soit Transport ([BJ87], [BSS90]).

3.2. Modes de défaillance et nombre de défaillances tolérées

Nous rappelons les définitions adoptées. Les processeurs sont sujets à différents modes de défaillance.

- Le mode le plus simple correspond à un **arrêt sur défaillance** : le processeur qui fonctionnait correctement s'arrête définitivement.
- Le deuxième mode de défaillance correspond à une **défaillance par omission**. Certains protocoles tolèrent uniquement les omissions en émission, d'autres tolèrent à la fois les omissions en émission et en réception (on parle alors d'omissions générales [GT89]). Le **degré d'omission** [DEL88] est défini comme étant le nombre maximum d'omissions consécutives, seuil au-delà duquel le processeur est définitivement défaillant.
- Le troisième mode concerne les **défaillances temporelles** : suite à une demande de service, le processeur fournit la réponse soit trop tôt, soit trop tard, soit jamais. Ce dernier cas correspond à une défaillance par omission.
- Le mode le plus complexe concerne les **défaillances arbitraires ou byzantines** ([LSP82], [CAS85]).

Selon les protocoles de diffusion, les défaillances du canal sont tolérées ou non. Dans la négative, elles sont imputées aux processeurs de communication. Dans l'affirmative elles sont du mode défaillance par omission. Elles peuvent conduire à un **partitionnement** toléré ou non par le protocole.

Par définition, la **vue du groupe** est l'ensemble des processeurs du groupe, autorisés à participer au protocole de diffusion.

Par abus de langage les processeurs membres de la vue du groupe sont dits **corrects.**

Un processeur **incorrect** est un processeur qui a été exclu de la vue du groupe suite à l'occurrence de défaillances.

La **résilience** est le nombre total de défaillances tolérées pour l'ensemble des processeurs de communication et du médium de communication. La résilience est un paramètre du protocole.

3.3. Description d'un protocole de diffusion atomique

La description d'un protocole de diffusion atomique comprend trois parties :

- le fonctionnement normal,
- la défaillance d'un émetteur,
- la défaillance d'un récepteur.

3.3.1. Le fonctionnement normal

Un protocole de diffusion atomique appartient soit à la classe des protocoles à contrôle centralisé, soit à la classe des protocoles à contrôle décentralisé ou réparti.

- un protocole de diffusion atomique **à contrôle centralisé** est basé sur l'existence d'un processeur privilégié unique (généralement appelé séquenceur), qui est chargé d'établir l'ordre total de remise des messages.

- dans un protocole de diffusion atomique **à contrôle décentralisé**, chaque processeur est susceptible d'interpréter ce rôle (exemple : séquenceur circulant, processeur élu, émetteur), et l'interprète de ce rôle varie au cours des diffusions (exemple : à chaque diffusion, toutes les k diffusions (k entier positif)).

- le protocole est dit **à validation symétrique** ([CAS85], [GT89]) si la décision de valider un message diffusé est prise symétriquement par l'émetteur et par les récepteurs : chacun décide de valider ou non un message selon les informations qu'il a reçues.

- pour les protocoles de **validation asymétrique** ([VRB89], [BJ87], [LG90] et [MA91]), l'émetteur seul est chargé de collecter les acquittements, qui lui permettent de décider seul de la validation du message. Cette décision est alors communiquée aux récepteurs.

La figure 3. classe les protocoles de diffusion atomique étudiés au paragraphe 4., selon ces deux caractéristiques. Nous rappelons qu'un tableau récapitulatif fait la synthèse de tous les protocoles de diffusion étudiés au cours de ce chapitre : les tableaux concernant les protocoles de diffusion atomique à contrôle centralisé (respectivement à contrôle décentralisé) sont donnés au paragraphe 4.3.(respectivement 4.4.).

Contrôle	centralisé		décentralisé	
Validation	symétrique	asymétrique	symétrique	asymétrique
Protocoles	**[GT89]**	**[KTH89]**	**[CM84]** **[CAS85]**	**[MA91]** **[BJ87]** **[BSS90]** **[GL90]** **[VRB89]**

figure 4. répartition des protocoles de diffusion atomique selon les classes contrôle et validation

Lors de la description du fonctionnement du protocole, nous mettons en évidence le nombre de phases nécessaires à la diffusion d'un message utilisateur, ainsi que le degré de concurrence. Par définition, **le degré de parallélisme** désigne en l'absence de défaillance, le nombre total de messages traités par toutes les invocations du protocole de diffusion, dont les exécutions se déroulent en parallèle.

3.3.2. La défaillance d'un émetteur

On distingue deux parties dans le traitement de la défaillance d'un émetteur :

- le sort du message dont l'émetteur est défaillant,
- l'exclusion de l'émetteur défaillant du groupe.

Certains protocoles ne traitent que la première partie se contentant de signaler à l'hôte l'existence d'un membre défaillant. C'est à l'hôte qu'il revient d'exclure ce membre du groupe. D'autres protocoles intègrent les mécanismes permettant d'exclure ce membre défaillant ([VRB89], [BJ87]). Quant au sort du message dont l'émetteur est défaillant, on distingue à nouveau deux types de protocoles. Les uns masquent la défaillance de l'émetteur [CRI85]. Les autres ont recours à un algorithme spécifique de détection/recouvrement. La détection est généralement décentralisée : chaque récepteur étant susceptible de détecter la défaillance de l'émetteur. Le recouvrement est exécuté soit par un processeur élu [VRB89], soit par un processeur prédéterminé [MA91], soit par concertation de tous les membres du groupe non défaillants [BJ87].

3.3.3. La défaillance d'un récepteur

La défaillance d'un récepteur peut prendre l'une des deux formes de recouvrement suivantes :

- dans la première forme de recouvrement, le protocole se contente de signaler à l'hôte la défaillance constatée. L'exclusion du membre défaillant est à la charge de l'hôte.

- dans la seconde forme de recouvrement, le protocole réalise lui-même l'exclusion du membre défaillant ([VRB89], [BJ87], et [MA91]).

3.4. Cinq propriétés d'un protocole de diffusion atomique

Les cinq propriétés que nous définissons dans ce paragraphe nous paraissent nécessaires dans un système réparti tolérant aux fautes.

3.4.1. Propriété 1 : Diffusion spontanée

Certains protocoles de diffusion atomique font hypothèse que chaque processeur de communication sait à priori à quel moment précis il a le droit de diffuser son message [GT89], cette information lui étant fournie par une horloge globale supposée ultra-fiable. D'autres protocoles, au contraire, **permettent à tout processeur de diffuser son message à tout instant** ([BJ87], [GL90], [VRB89] et [MA91]). **De tels protocoles gèrent les diffusions spontanées.**

3.4.2. Propriété 2 : Diffusions concurrentes

Certains protocoles de diffusion atomique ne considèrent pas les diffusions concurrentes [GT89] : seul un processeur est autorisé à diffuser un message vers le groupe de diffusion. D'autres au contraire, autorisent simultanément plusieurs processeurs à diffuser leur message. Il faut alors déterminer un ordre total de remise de ces messages concurrents. Si la propriété 1 est satisfaite et si les processeurs peuvent défaillir par omission, alors il est nécessaire que le protocole de diffusion atomique soit capable de gérer les diffusions concurrentes. Par exemple, dans la figure 5., *P1* diffuse son message de numéro de séquence *m*, estampillé *<m,P1>*. P2 sur réception du message de *P1*, acquitte. *P3* diffuse son message avec le même numéro de séquence *m* que celui utilisé par *P1* du fait qu'il a omis de recevoir le message de *P1*. Son message est estampillé *<m,P3>*. Sur non réception de l'acquittement de *P3*, *P1* rediffuse son message. *P3* reçoit ce message et acquitte *P1*. Par conséquent, l'ordre de réception des messages chez *P1* et *P2* est identique (*<m, P1> puis <m, P3>*), alors que chez *P3* cet ordre est (*<m, P3> puis <m, P1>*). Quelque soit l'ordre de réception, tous les processeurs doivent remettre à leur hôte, ces deux messages dans le même ordre. Cet ordre est déterminé par le protocole de diffusion.

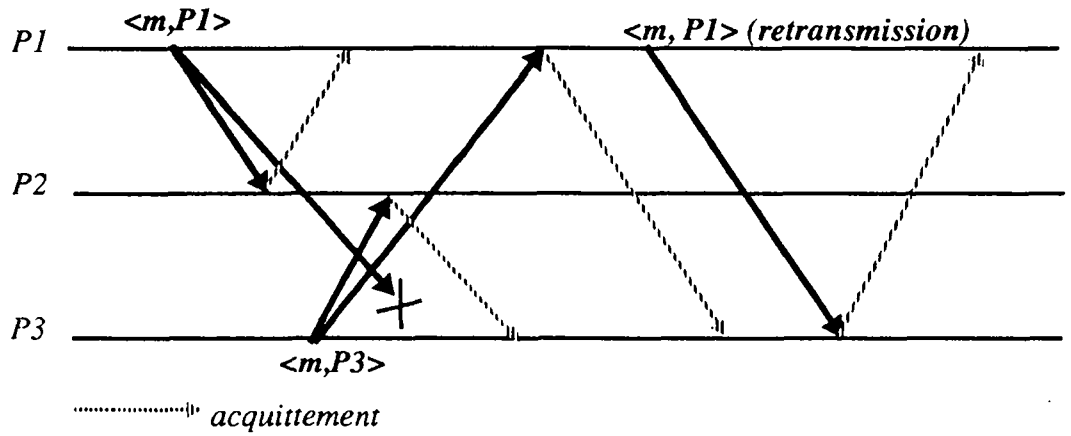

figure 5. diffusions concurrentes

3.4.3. Propriété 3 : Nombre minimum d'annulation de messages

Dans certains protocoles, si un émetteur défaille en cours de diffusion, alors la diffusion est annulée. C'est donc l'aspect terminaison au plus tôt qui est privilégié, au détriment de l'aspect terminaison avec succès (remise du message aux processeurs d'application). Dans d'autres protocoles, l'annulation d'une diffusion n'intervient que lorsque l'émetteur est défaillant et aucun processeur du groupe ne peut rediffuser le message dans le groupe. C'est l'aspect terminaison avec succès qui est privilégié par ces protocoles. De ce fait, ils minimisent le nombre d'annulations de messages (exemple [MA91]).

3.4.4. Propriété 4 : Pas d'exclusion abusive du groupe de diffusion

Dans un système réparti faisant hypothèse de défaillances par omission pour les processeurs de communication et le médium de communication, un point important à considérer est le critère utilisé pour exclure du groupe de diffusion un membre silencieux. Certains protocoles autorisent l'émetteur, n'ayant pas reçu l'acquittement d'un récepteur, à exclure purement et simplement ce dernier. Cela peut conduire à la situation paradoxale suivante : un membre fautif exclut successivement les membres non fautifs de son groupe. Or, rappelons qu'il existe trois raisons possibles pour que *P1* ne reçoive pas un message de *P2* : soit *P2* a omis d'émettre, soit le canal de diffusion a eu une omission, soit *P1* a omis de recevoir. **Par définition, un protocole de diffusion atomique évite toute exclusion abusive si la décision d'exclure un processeur silencieux résulte d'un consensus majoritaire au sein du groupe de diffusion.**

3.4.5. Propriété 5 : Pas de contamination du système

Lors de notre étude, nous avons constaté que deux types de comportements pouvaient contaminer un système. **Le premier type de contamination peut être provoqué par le comportement d'un processeur exclu de la vue du groupe** : il faut éviter que les messages émis par ce processeur incorrect soient pris en compte par les autres processeurs du groupe de diffusion [GT91]. Par définition, un protocole de diffusion atomique évite la contamination du premier type s'il satisfait les conditions suivantes :

- **c1) le message diffusé par un processeur incorrect est rejeté par tous les processeurs qu'ils soient eux même corrects ou incorrects,**

- **c2) et la séquence des messages délivrés par un processeur incorrect est un préfixe de la séquence des messages délivrés par les processeurs corrects.** En d'autres termes, il n'existe aucun message délivré par un incorrect qui ait été annulé par les processeurs corrects et inversement. Cette condition permet d'assurer qu'aucun message validé par au moins un processeur n'est annulé par les autres processeurs du groupe. Considérons par exemple, le protocole suivant : l'émetteur diffuse la validation d'un message *m*, confirme la validation localement, puis défaille. Aucun processeur ne reçoit cette validation. La décision d'annuler *m* est alors prise à l'unanimité. Ceci est un exemple de protocole où un message validé par certains processeurs, est annulé par les autres.

Dans un système transactionnel, la condition c2) est nécessaire pour garantir l'atomicité des transactions. De plus, elle simplifie considérablement la procédure d'insertion : le processeur qui s'insère doit simplement rattraper son retard en exécutant et validant les transactions validées en son absence ; il ne doit pas demander un télé-chargement complet de ces données.

Le second type de contamination peut quant à lui être dû à une violation des hypothèses de bornes supposées par le protocole. Citons à titre d'exemples, le nombre maximum d'omissions d'un processeur avant arrêt, l'écart maximum entre deux horloges synchronisées, le nombre maximum de défaillances tolérées par le protocole.

Par définition, un protocole de diffusion atomique évite la contamination du second type s'il est capable de détecter en cours de fonctionnement une telle violation (par exemple, dans [MA91] chaque processeur doit vérifier qu'il est capable de communiquer avec une majorité de processeurs corrects, avant d'être autorisé à prendre une décision).

4. Protocoles de diffusion atomique

4.1. Protocole de diffusion atomique à contrôle centralisé

Un processeur unique *P0* est initialement choisi pour jouer le rôle de séquenceur : il détermine l'ordre total de remise des messages. Un processeur qui veut diffuser un message l'envoie à *P0*. L'ordre d'arrivée des messages à *P0* crée un ordre total sur ces messages. Le processeur *P0* diffuse les messages reçus dans l'ordre où il les a reçus. Ce type de protocole est assez simple, mais présente deux inconvénients majeurs : la défaillance du séquenceur et le goulot d'étranglement qu'il constitue. En ce qui concerne la défaillance de *P0* il est nécessaire d'adjoindre un algorithme d'élection pour désigner un nouveau séquenceur et de récupérer les messages que *P0* a reçus mais n'a pas encore diffusés à tous les récepteurs. Le goulot d'étranglement est quant à lui intrinsèque au principe choisi.

Nous étudions tout d'abord un protocole supposant que les processeurs sont de type arrêt sur défaillance.

4.1.1. Processeur à arrêt sur défaillance

4.1.1.1. Protocole 4 : An efficient reliable broadcast protocol

F. Kaashoek, S. Tanenbaum, F. Hummel, E. Bal, "An efficient reliable broadcast protocol" [KTH].

4.1.1.1.1. Modèle

4.1.1.1.1.1. Architecture du système

Les processeurs de communication sont connectés à un bus à diffusion. Les délais de transmission sont supposés bornés. Les processeurs supportent le protocole de diffusion fiable décrit dans ce paragraphe. Le système ne possède pas d'horloge globale mais chaque processeur de communication a accès à une horloge locale.

4.1.1.1.1.2. Mode de défaillance

- **Mode de défaillance des processeurs** est de type arrêt sur défaillance ou fail-stop.
- **Mode de défaillance du médium de communication** : le bus à diffusion est sujet aux omissions (perte de messages), mais possède un degré de redondance tel que le partitionnement du système est impossible.

4.1.1.1.3. Propriétés caractéristiques

Le protocole de diffusion atomique, garantit les trois propriétés de **Consensus unanime, d'Ordre total** et de **Terminaison**.

4.1.1.1.2. Description du protocole de diffusion

4.1.1.1.2.1. Principe

Ce protocole utilise un serveur de séquencement (ou séquenceur), sur un processeur particulier, pour assurer la mise en ordre des messages. Pour diffuser un message, l'émetteur l'envoie au séquenceur en point-à-point, en lui fournissant le numéro du dernier message reçu et ceci après s'être assuré que sa diffusion précédente était terminée. Le séquenceur attribue un numéro d'ordre et le diffuse (on suppose que le message s'adresse à tous les processeurs du réseau et non à un sous-ensemble). L'émetteur se bloque jusqu'à ce qu'il reçoive l'estampille attribuée par le séquenceur (pour éviter d'être bloqué indéfiniment, il arme un réveil de retransmission, qui sur déclenchement l'autorise à réémettre un double de son message au séquenceur). Le séquenceur conserve une copie des messages qu'il a diffusés, avec leur numéro d'ordre, dans un tampon "historique". Si tous les messages jusqu'à un numéro d'ordre k ont été acquittés par tous les destinataires, ils peuvent être éliminés du tampon. Ce protocole fait donc partie des protocoles à validation asymétrique (cf paragraphe 3.3.1.).

Chaque destinataire accepte les messages dans l'ordre de leur numéro et les délivre dans cet ordre. De plus il conserve le numéro s du dernier message reçu. Si un message arrive avec un numéro supérieur à $s+1$, le récepteur le met en attente et demande au séquenceur de lui renvoyer une copie du ou des messages manquants. Le séquenceur retrouve ces messages dans son tampon historique.

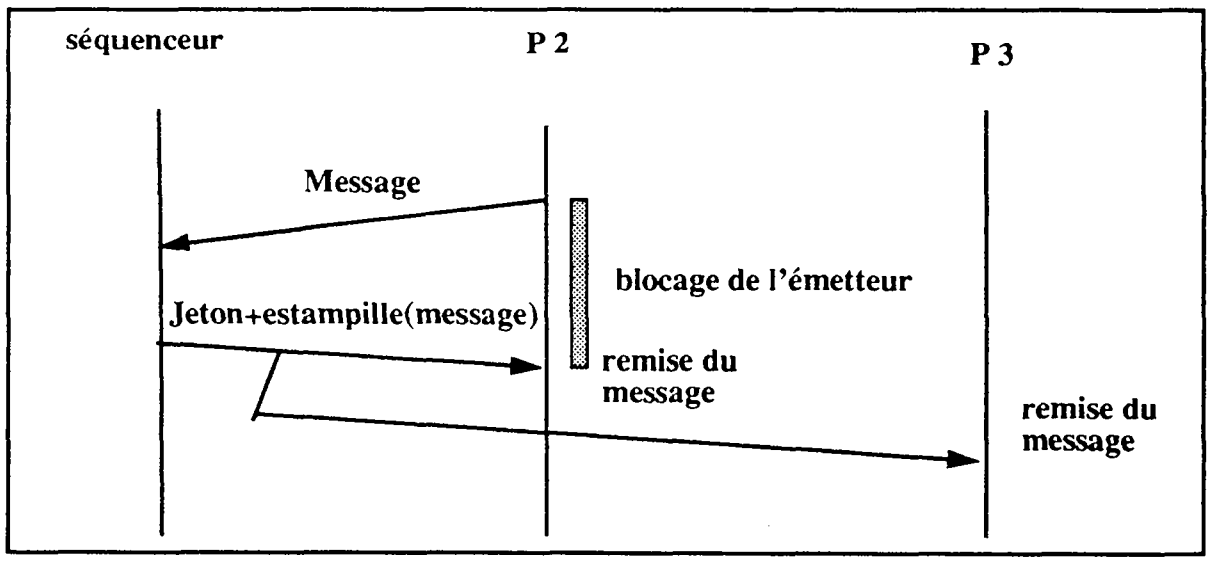

figure 6. Exemple de diffusion

Le séquenceur conserve localement, dans un buffer, les k derniers messages qu'il a estampillés. Le séquenceur retire un message m de son tampon, dès qu'il a reçu la preuve que tous les processeurs ont bien reçu ce message (sur réception d'un message de vie ou d'un message utilisateur). Si le tampon historique devient plein, le séquenceur lance un protocole de synchronisation avec lequel il s'assure que tous les messages présents dans le tampon ont bien été reçus par tous les destinataires. Il renvoie si nécessaire les messages qui ont été perdus, puis il vide le tampon.

4.1.1.1.2.2. Défaillance d'un processeur

Toutes les x réceptions de messages diffusés (x est une constante du protocole), chaque processeur envoie au séquenceur un message de vie indiquant le numéro de séquence du dernier message reçu. Ceci permet donc au séquenceur de détecter la défaillance d'un processeur (émetteur ou récepteur).

4.1.1.1.2.3. Protocole de synchronisation

Le protocole de synchronisation s'exécute en deux phases, ce qui permet de garantir que tous les processeurs du groupe de diffusion reçoivent tous les messages contenus dans la mémoire tampon du séquenceur. Ce protocole nécessite que le trafic soit suspendu entre les processeurs : au cours de ces deux phases, un processeur n'est pas autorisé à diffuser un nouveau message (si tel est le cas, le séquenceur refuse d'estampiller tout nouveau message).

Dans la première phase, le séquenceur diffuse un message d'invitation contenant le dernier numéro de séquence utilisé. Sur réception de ce message un récepteur vérifie qu'il ne lui manque pas de messages. S'il est à-jour il acquitte le séquenceur sinon il lui redemande tous les messages manquants. A la fin de la phase 1 du protocole de synchronisation, tous les récepteurs ont reçu tous les messages diffusés, permettant ainsi au séquenceur de vider son tampon.

Dans la seconde phase, le séquenceur diffuse un message de validation informant tous les membres du groupe que tous les processeurs sont à-jour. Sur réception d'un message de validation, un récepteur envoie un acquittement de validation et reprend le cours normal du protocole de diffusion fiable. Le séquenceur reprend le cours normal du protocole de diffusion fiable lorsqu'il a reçu les acquittements de tous les processeurs.

4.1.1.1.2.4. Défaillance du séquenceur

Quand un processeur n'obtient pas de réponse du séquenceur (exemple : suite à une demande de retransmission de messages manquants et ceci en dépit de plusieurs essais successifs), il suppose que le séquenceur est défaillant ; un protocole d'élection d'un nouveau séquenceur est alors exécuté. Ce protocole n'est pas spécifié dans l'article. Ce protocole doit assurer l'unicité du séquenceur, et garantir la non perte de messages déjà remis par certains processeurs. Il doit également éviter la formation de partitions évoluant en parallèle.

4.1.1.1.3. A propos des cinq propriétés

4.1.1.1.3.1. Diffusion spontanée

Les diffusions spontanées sont autorisées par le protocole. Il existe une limitation empêchant toute anticipation d'émission : un émetteur est autorisé à envoyer son message au séquenceur uniquement lorsque sa diffusion précédente est terminée.

4.1.1.1.3.2. Diffusions concurrentes

Dans ce protocole, comme dans tous les autres protocoles utilisant le principe de processeur central, responsable de l'ordonnancement des messages (estampillage), on ne peut pas parler à proprement dit de diffusions concurrentes.

La remise d'un message est effectuée suivant l'ordre croissant des estampilles affectées aux messages qui est en fait l'ordre de réception de ces messages par le processeur central ou séquenceur.

4.1.1.1.3.3. Nombre minimal d'annulation de messages

Un message *m* est annulé dans deux cas :

- 1) l'émetteur de *m* est défaillant et le séquenceur n'a pu recevoir *m* suite à une omission du réseau.
- 2) l'émetteur du message et le séquenceur ne font qu'un, le séquenceur défaille et aucun processeur n'a reçu le message.

4.1.1.1.3.4. Pas d'exclusion abusive

Dans ce protocole, ce protocole suppose qu'un processeur qui ne répond plus au séquenceur, malgré plusieurs tentatives de ce dernier, est définitivement défaillant et donc ne fait plus partie du groupe. Ainsi, l'exclusion d'un processeur est relative à un seul avis et non à un consensus majoritaire.

Dès que le séquenceur ne communique plus avec un processeur du groupe ce dernier est exclu du groupe, ceci pouvant conduire à des exclusions abusives, car le canal de communication est sujet à des omissions.

4.1.1.1.3.5. Non contamination du système

Le séquenceur ignore les messages d'un processeur exclu. Par contre lors de la défaillance du séquenceur, la procédure d'élection du nouveau séquenceur doit assurer qu'aucune contamination du système n'est possible (exemple : la perte de messages validés), cette procédure n'étant pas décrite dans cet article, on ne peut pas se prononcer sur la propriété de non-contamination du système.

4.1.1.1.4. Conclusion

Le protocole peut être rendu tolérant à une défaillance du séquenceur par l'élection d'un nouveau séquenceur. Le protocole d'élection, non décrit dans la description faite par l'auteur peut, par exemple, choisir le processeur dont le dernier numéro de séquence est le plus élévé. Cependant, si l'on veut garantir que le protocole ne perd aucun message déjà remis par certains processeurs et ce en dépit de défaillance de processeurs il faut revoir complètement le fonctionnement du protocole, afin d'interdire la remise d'un message tant qu'il n'a pas été acquitté par au moins $r+1$ processeurs, r caractérisant la résilience du protocole.

Enfin, un inconvénient propre à cette catégorie de protocole est la possibilité d'avoir un goulot d'étranglement au niveau du séquenceur. Notons que le blocage du système lors du protocole de synchronisation est regrettable.

4.1.2. Processeur à omission sur défaillance

4.1.2.1. Protocole 5 : Reliable Broadcast in Synchronous and Asynchronous environment

A. Gopal, S. Toueg, "Reliable Broadcast in Synchronous and Asynchronous environment" [GT89].

4.1.2.1.1. Modèle

4.1.2.1.1.1. Modèle d'architecture

Le protocole de diffusion fiable permet à un groupe de *n* processeurs, totalement connectés en point-à-point, d'échanger des messages. Ces processeurs ne partagent pas de mémoire commune mais ont tous accès à une horloge globale fiable. Les délais de transmission d'un message sur le médium de communication sont bornés (protocole synchrone).

Une diffusion fiable est réalisée entre un processeur central, le ***transmitter*** (unique processeur autorisé à émettre des message à travers le groupe) et les récepteurs.

4.1.2.1.1.2. Mode de défaillance

- **mode de défaillance des processeurs** : la défaillance d'un processeur est de type **omission**. Quand un processeur commet des fautes d'omission il est dit **fautif**, et au cours d'une diffusion, seuls *t* processeurs peuvent être **fautifs**.
- **mode de défaillance du médium de communication** : le médium de communication est **fiable** : pas d'omission et pas de partionnement possible.

4.1.2.1.1.3. Propriétés caractéristiques

Le *transmitter* diffuse une séquence de messages estampillés par leur numéro de séquence. Le message de numéro de séquence k est appelé $k^{ième}$. Chaque récepteur, y compris le *transmitter*, doit remettre cette même séquence de messages à son hôte.
Le protocole garantit les propriétés suivantes :

- **validité** : si le *transmitter* est correct, alors quelque soit $k>0$, chaque processeur correct remet à son hôte, les *k* premiers messages diffusés par le transmitter ,
- **uniformité** : si un processeur remet un $k^{ième}$ message, chaque processeur correct remet à terme, ce même $k^{ième}$ message. De plus ce message est soit le $k^{ième}$ message diffusé par le *transmitter* soit un message nul (uniquement dans le cas où le *transmitter* est défaillant).
- **ordre total** : cet ordre, imposé par l'unique source de messages (le *transmitter*), est respecté par chaque récepteur.

4.1.2.1.2. Description du protocole

4.1.2.1.2.1. En l'absence de défaillance du transmitter

Le protocole proposé par Ajei Gopal et Sam Toueg se déroule en **rounds successifs**. Le terme de round est préféré à celui de phase lorsqu'une synchronisation **par l'horloge globale** commune à tous les processeurs est effectuée (tous les processeurs connaissent le numéro du round en cours d'exécution).

Pour effectuer une diffusion fiable, au minimum trois rounds sont nécessaires : seul le premier est à l'initiative du *transmitter*.

Supposons que le round exécuté soit le $k^{ième}$. Au cours de ce round k, le *transmitter* diffuse par *(SEND-INIT,k,m_k)* son $k^{ième}$ message noté m_k . Tous les processeurs doivent recevoir ce message au cours de ce round k (en l'absence de fautes). Au round suivant, le $(k+1)^{ième}$, un processeur *P*, différent du *transmitter*, diffuse la valeur reçue dans un message de type *(SEND-ECHO:k+1,m_k)*, sous réserve qu'il ne considère pas le *transmitter* fautif. A la fin de ce même round, si *P* a reçu au moins un message de type *(SEND-ECHO:k+1,m_k)*, il accepte le message *mk*, mais il ne le remet pas encore. Le transmitter diffuse *(SEND-INIT:k+1,m_{k+1})*. Au $(k+2)^{ième}$, lorsque *P* reçoit un nouveau message *(SEND-ECHO:k+2,m_{k+1})*, émis par un processeur différent de *P*, alors *P* remet le message m_k à son hôte.

La décision de valider un message (dans ce protocole synonyme de remettre)est prise symétriquement par l'émetteur et par les récepteurs, par conséquent ce protocole appartient à la classe des protocoles dits à validation symétrique.

Tant que le *transmitter* n'est pas défaillant, il diffuse consécutivement ses messages, qui sont acceptés, puis remis par tous les processeurs dans l'ordre dans lequel ils ont été envoyés. La figure 7., décrit les rounds *k, k+1 et k+2*.

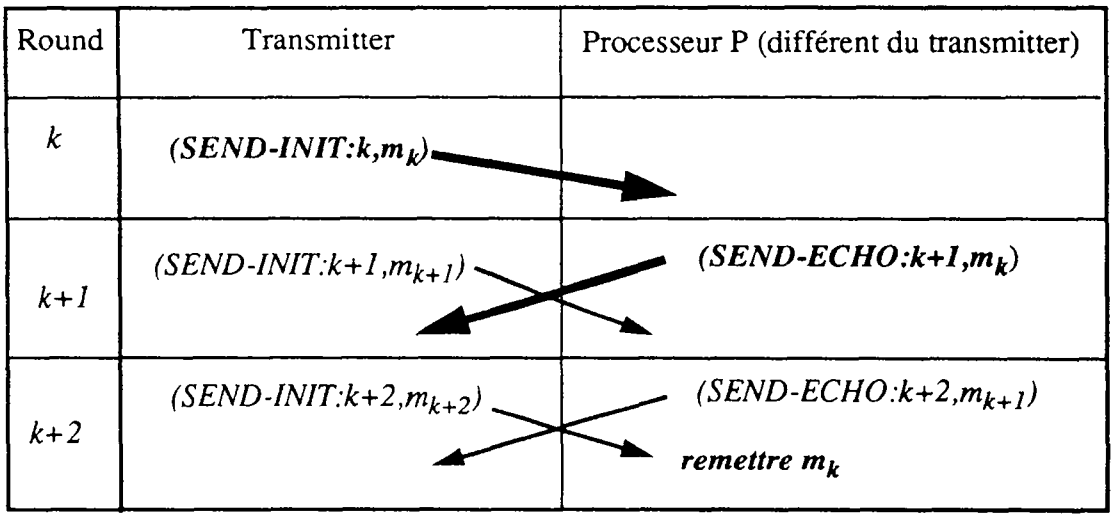

figure 7. Description de trois rounds successifs

4.1.2.1.2.2. Défaillance du transmitter

Lorsqu'à la fin d'un round *k*, un processeur *P* s'aperçoit que le *transmitter* ne communique pas avec *n-t* processeurs, il le déclare fautif. *P* déduit cette information des messages qu'il a reçus des autres processeurs : un processeur n'ayant rien reçu du *transmitter* ou considérant le *transmitter* comme fautif, diffuse un message de type *(SEND-ECHO:k,F)* à la place de *(SEND-ECHO:k,m_k)*, ce qui permet à *P* de connaître l'état "global" du *transmitter* dans le groupe.

Au round suivant, le $(k+1)^{ième}$, *P* invoque un **protocole de consensus** afin d'obtenir un consensus parmi tous les processeurs non défaillants sur les deux derniers messages diffusés par le *transmitter* avant sa défaillance.

Lorsque le recouvrement est terminé, le système se bloque : il n'y a pas d'élection d'un nouveau *transmitter*.

4.1.2.1.2.3. Défaillance d'un processeur

Lorsqu'à la fin d'un round k, un processeur P s'aperçoit qu'il ne communique pas avec $n-t$ processeurs, il se considère comme fautif. Il se suicide, mais n'en avertit ni le *transmitter* ni les autres membres du groupe.

4.1.2.1.2.4. Protocole de consensus

Le protocole de consensus est invoqué, simultanément, par tous les processeurs P détectant la défaillance du *transmitter*. Ce protocole permet d'obtenir un consensus sur les messages diffusés par le *transmitter* lors des trois rounds précédant la détection de sa défaillance.

Si le nombre de fautes tolérées par le protocole de diffusion fiable est de t, alors $t+3$ rounds sont nécessaires. Une description sommaire des invocations de ce protocole est donnée ci-dessous:

Un processeur P exécute ce protocole au round $r+1$, si au cours du round r il n'accepte pas le message m_{r-1} du *transmitter* (fautif).

- P doit avoir accepté et remis m_{r-3}, mais il est possible que certains processeurs ne l'aient pas remis. Donc au round $r+1$, P rediffuse le message m_{r-3}. Le consensus sur ce message se termine au round $r+t$ avec deux résultats possibles : soit m_{r-3}, soit F (cas où moins de $(n-t)$ processeurs ont participé au consensus). Dans les deux cas P ayant déjà remis m_{r-3} ne le remet pas à nouveau.
- P doit avoir accepté m_{r-2}, mais il est possible qu'un autre processeur ait déjà accepté et remis m_{r-2}. Donc, au round $r+2$, P rediffuse le message m_{r-2}. Le consensus sur m_{r-2} se termine au round $r+t+1$; si le résultat est m_{r-2}, alors P remet m_{r-2} .
- P n'a pas accepté le message diffusé au round $r-1$, cependant il est possible qu'un autre processeur ait accepté ce message. Donc, au round $r+3$, P diffuse le message qui a été diffusé au round $r-1$. Ce protocole se termine au round $r+t+2$; si le résultat est m_{r-1} alors P remet m_{r-1} .

La manière dont est obtenu le consensus sur la valeur d'un message n'est pas décrite dans cet article. On sait uniquement qu'il nécessite t rounds pour chaque message.

4.1.2.1.3. A propos des 5 propriétés

4.1.2.1.3.1. Diffusions spontanées

Ce protocole de diffusion fiable est cadencé par une horloge globale, délimitant les rounds. Une diffusion ne peut être initialisée (par le *transmitter*) qu'au début d'un round. Donc la propriété de diffusion spontanée n'est pas satisfaite.

4.1.2.1.3.2. Diffusions concurrentes

Seul le *transmitter* joue le rôle d'émetteur dans le groupe, ce qui lui permet d'imposer l'ordre de remise des messages. Ce protocole ne satisfait donc pas la propriété de diffusions concurrentes.

4.1.2.1.3.3. Nombre minimal d'annulation de messages

L'annulation d'un message est effectuée suite à la défaillance du *transmitter*, et à la non réception du message par $n-t$ processeurs, si t désigne le nombre de fautes tolérées par le protocole.

4.1.2.1.3.4. Pas d'exclusion abusive

L'exclusion d'un processeur est réalisée soit suite à un départ volontaire (le processeur appartenant à moins de *(n-t)* s'exclut de lui-même), soit suite à au moins *(n-t)* avis : *(n-t)* processeurs détecte la défaillance du *transmitter*. Pour que le protocole satisfasse la propriété : pas d'exclusion abusive, il est nécessaire que t soit inférieur ou égale à la majorité du groupe moins un.

4.1.2.1.3.5. Pas de contamination du système

En cours de fonctionnement, les processeurs vérifient d'une part leur aptitude à communiquer avec le restant du groupe, et d'autre part, l'aptitude du *transmitter* à communiquer. En effet, dès qu'un processeur reçoit moins de (*n-t*) réponses, il le détecte et se suicide. Si le nombre de défaillances de processeurs dépasse *t*, chaque processeur le détecte et se suicide. La propriété de non contamination est garantie si *t* est inférieur à la majorité, ceci afin d'éviter la présence de plusieurs partitions actives au sein du système.

4.1.2.1.4. Conclusion

La fiabilité de l'horloge centrale constitue le point faible de ce protocole. Le protocole exposé ne traite le cas que d'un *transmitter*.

Comment un processeur différent du *transmitter* peut-il diffuser son message ? Doit-il s'adresser au *transmitter* ou devient-il lui-même *transmitter*.

Sur défaillance du *transmitter*, le système s'arrête.

4.2. Récapitulatif des protocoles de diffusion atomique à contrôle centralisé

PROTOCOLE DE DIFFUSION ATOMIQUE	KAASHOEK - TANENBAUM	GOPAL-TOUEG
Validation	asymétrique	symétrique
Nombre de phases	deux phases	3 rounds
mode de défaillance	arrêt sur défaillance	omission par défaillance
Horloge	locale	globale fiable
Degré de parallélisme	n en 1ère phase 1 en 2ème phase	1 en 1er round 1 en 2ème round 1 en 3ème round
partitionnement	non toléré	non toléré
algorithme de recouvrement	le protocole de recouvrement n'est pas décrit dans l'article	(t+3) rounds avec t= nombre de défaillances tolérées
Les 5 propriétés		
Diffusion spontanée	oui sous réserve que le message précédent soit validé	non
Diffusion concurrente	non	non
Nombre min. d'annulations	2 cas d'annulation pour un message 1) l'émetteur est défaillant et le séquenceur n'a pas reçu le message 2) l'émetteur et le séquenceur ne font qu'un, défaillent et aucun n'a reçu le message	annulation d'un message si défaillance du transmitter et réception par moins de *n-t* processeurs
Non contamination	la contamination dépend de l'algorithme d'élection du séquenceur	pas de contamination si vérification en cours de fonctionnement qu'il existe moins de t défaillances, t<majorité
Exclusion abusive	exclusion abusive possible : 1 seul avis est suffisant pour exclure un membre du groupe	pas d'exclusion abusive : il est nécessaire d'avoir *n-t* avis avec *n* = nbre de processeurs du groupe

figure 8. Diffusion atomique - contrôle centralisé - arrêt et omission

4.3. Protocole de diffusion atomique à contrôle décentralisé

Les protocoles présentés ci-après contrôlent de manière décentralisée les diffusions. Nous classons ces protocoles selon le mode de défaillance qu'ils tolèrent.

4.3.1. Processeur à arrêt sur défaillance

4.3.1.1. Protocole 3 : Reliable Broadcast protocols

J. Chang and N. Maxemchuck, "Reliable Broadcast protocols"[CM84].

4.3.1.1.1. Modèle

4.3.1.1.1.1. Architecture du système

Le protocole de diffusion fiable permet à un groupe de processeurs, connectés à un bus à diffusion, d'échanger des messages.

4.3.1.1.1.2. Mode de défaillance

Le mode de défaillance des processeurs est de type **arrêt sur défaillance** : les fautes d'omission ne sont pas tolérées. Seul le médium de communication est sujet aux **fautes d'omission**.

Il y a **défaillance** lorsque deux processeurs ne peuvent pas communiquer entre eux et ceci après R retransmissions successives. Un processeur est dit **non-opérationnel** s'il ne répond pas aux sollicitations d'un autre processeur. Il doit alors avoir recours à une procédure de recouvrement pour se réinsérer dans le groupe de diffusion et pour devenir à nouveau **opérationnel**.

4.3.1.1.1.3. Propriétés caractéristiques

Ce protocole de diffusion est un protocole de diffusion **atomique**. Il garantit l'unanimité, l'ordre total, et la terminaison.

4.3.1.1.2. Description du protocole

4.3.1.1.2.1. Principe

Le protocole de Chang et Maxemchuck utilise la technique du jeton circulant sur un anneau virtuel. Tous les processeurs participant à une conversation constituent un anneau virtuel. Ils peuvent diffuser leur message à tout moment. Les processeurs participants insèrent les messages reçus dans une file de messages reçus (en attente d'être estampillés), mais n'acquittent pas le message. S'ils détectent des messages ou des acquittements manquants, ils en font la demande au processeur-jeton courant. Seul le processeur possédant le jeton est autorisé à estampiller des messages. Les messages sont délivrés dans l'ordre fourni par leur estampille.

Afin de limiter la vulnérabilité du processeur-jeton, le rôle priviligié du processeur-jeton est réparti équitablement dans le temps entre les différents processeurs participants : jeton circulant. Outre l'estampillage des messages, le processeur-jeton doit également retransmettre des messages estampillés aux processeurs qui en ont fait la demande, et assurer le transfert du jeton au nouveau processeur-jeton.

Seul le processeur-jeton doit fournir un message d'acquittement à l'émetteur. Cet acquittement permet d'une part de diffuser l'estampille qu'il a affectée au message et d'autre part de passer le jeton au prochain processeur-jeton (successeur de l'actuel processeur-jeton sur l'anneau virtuel). Sur réception de cet acquittement, le destinataire du jeton, vérifie qu'il peut accepter le jeton : ce processeur doit avoir reçu tous les messages qui ont été jusqu'à présent estampillés. Si ce n'est pas le cas, il fait une demande de retransmission au processeur-jeton. Dès qu'il est à jour, il est autorisé à devenir le nouveau processeur-jeton et par conséquent, est autorisé à estampiller et acquitter les messages reçus, ou bien à faire tourner le jeton pour valider un message. Le passage du jeton est considéré comme effectif, lorsque l'ancien processeur-jeton a reçu la confirmation du nouveau processeur-jeton.

Dans le cas où le processeur-jeton n'a aucun message à acquitter ou à valider, il diffuse une confirmation mais dans ce cas ne fait tourner le jeton qu'après déclenchement d'un temporisateur. Dès qu'un message a été estampillé et qu'il a été suivi de L passages de jeton, le processeur-jeton courant le valide. Ainsi, L passages de jetons consécutifs sont nécessaires pour valider un message estampillé. Le paramètre L désigne le degré de résilience du protocole.

Cette procédure n'assure pas que tous les processeurs opérationnels ont validé le message, car seuls les processeurs ayant détecté L passages de jeton depuis son estampillage, valident ce message. Ainsi, afin d'assurer la propriété de consensus unanime, il est nécessaire de faire tourner le jeton jusqu'à ce que le processeur-jeton responsable de la validation reçoive de nouveau le jeton.

Par voie de conséquence, le coût en nombre de messages devient important : $(N+L)$ passages de jeton sont nécessaires pour assurer que les N processeurs opérationnels appartenant au groupe ont validé le message.

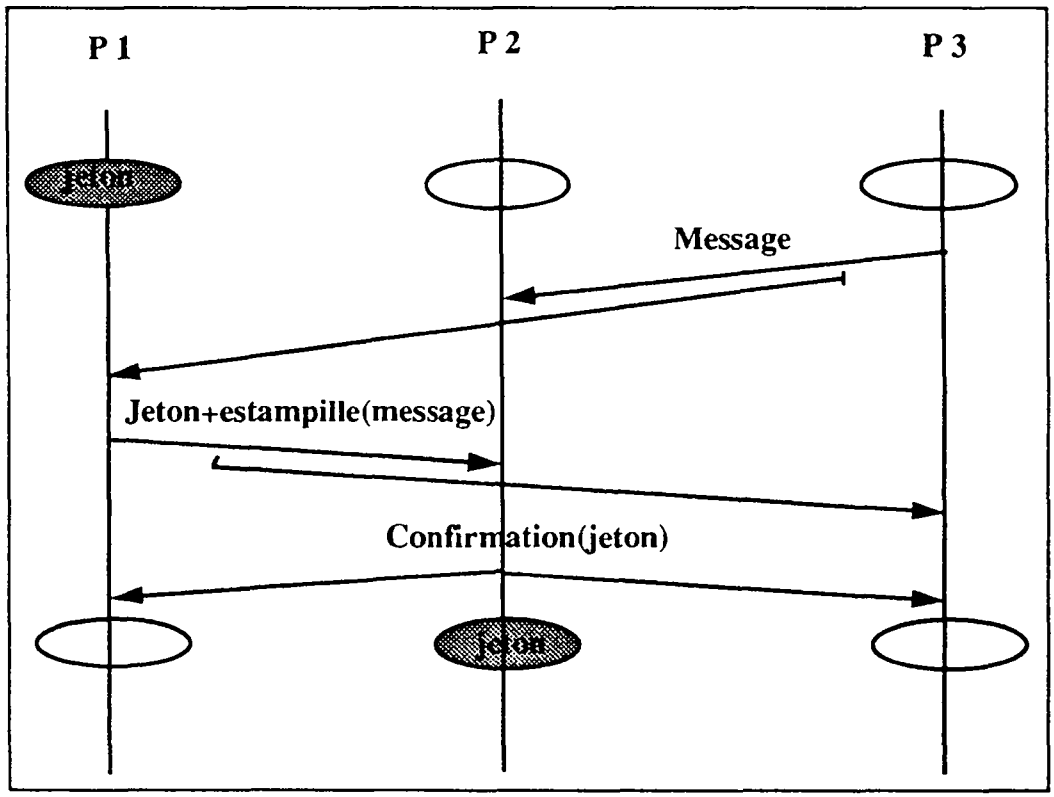

figure 9. exemple de diffusion

4.3.1.1.2.2. Détection de la défaillance d'un processeur

4.3.1.1.2.2.1. Détection de la défaillance du processeur-jeton

La détection de la défaillance du processeur-jeton est réalisée :

- par un émetteur : sur non réception de l'acquittement du processeur-jeton après R rediffusions successives de son message, un émetteur initialise la procédure de reconfiguration.
- par un récepteur : un récepteur ne recevant pas tous les messages estampillés manquants qu'il a demandés, initialise la procédure de reconfiguration.

4.3.1.1.2.2.2. Détection de la défaillance d'un récepteur

La détection de la défaillance d'un récepteur ne peut se faire que par le processeur-jeton et ne peut concerner qu'un seul processeur : il s'agit du prochain processeur-jeton appartenant à la liste jeton. Il est donc tout-à-fait possible que la détection de la défaillance d'un processeur n'intervienne qu'après une rotation complète du jeton.

Le processeur-jeton initialise la procédure de reconfiguration.

4.3.1.1.2.3. Traitement des défaillances : protocole de reconfiguration

Le protocole de reconfiguration est déclenché quand un processeur détecte une défaillance. Ce processeur est appelé *initiateur*. Un processeur ne peut être *initiateur* que s'il appartient à l'anneau virtuel. Lorsque plusieurs processeurs détectent la défaillance simultanément, il peut y avoir plusieurs *initiateurs*.

Le protocole doit conduire d'une part à la formation d'un nouvel anneau virtuel (donc à la construction d'une nouvelle version de la liste des processeurs participants), et d'autre part à l'élection d'un nouveau processeur-jeton, si nécessaire.

Il doit en outre assurer : 1) l'unicité de la liste des processeurs et du site-jeton ; 2) la cohérence de l'estampille attendue par chaque processeur appartenant à la liste ; 3) l'absence de perte de messages déjà acquittés.

Pour cela, ce protocole est exécuté en deux phases qui correspondent à la création de la nouvelle liste des processeurs et le test d'unicité de cette liste (première phase), et à l'élection du nouveau processeur-jeton (seconde phase).

4.3.1.1.2.3.1. Phase 1 : Création et validation de la nouvelle liste des processeurs

- Création de la liste des processeurs : le début de cette phase est marqué par la diffusion d'un message d'invitation par un *initiateur*. Ce message d'invitation contient le numéro de la nouvelle version de la liste à créer et le numéro de l'*initiateur*. Comme il peut y avoir plusieurs *initiateurs*, il peut y avoir plusieurs messages d'invitation. Un processeur désirant participer à la conversation doit répondre par un acquittement positif à l'une des invitations reçues. Une liste des processeurs participants est reconstruite par chaque *initiateur* à partir des acquittements reçus. L'existence de listes concurrentes étant tout à fait possible, il est nécessaire d'introduire un mécanisme de validation afin de n'en retenir qu'une seule.

- la validation de la liste des processeurs est effective lorsque les deux tests suivants sont satisfaits :

  - test de consensus majoritaire : une nouvelle liste ne peut être validée que lorsqu'elle contient une majorité de processeurs ayant appartenu à la liste précédente. Ce critère garantit l'unicité de la nouvelle liste.

• test de séquence des numéros de version : tout processeur ne doit rejoindre une liste que si celle-ci possède un numéro de version supérieur au numéro de la liste détenue par le processeur. On peut remarquer, que la résolution du conflit entre plusieurs *initiateurs* lors du premier test (un *initiateur* n'ayant pu constituer une majorité renonce à établir sa propre liste et rejoint la liste qui a obtenu la majorité) nécessite que tous les processeurs aient connaissance de la situation des différents *initiateurs*.

Une fois la liste validée, l'*initiateur* diffuse cette liste à travers le groupe et génère un nouveau jeton.

4.3.1.1.2.3.2. Seconde phase : élection du nouveau processeur jeton

Une fois la liste validée, se pose le problème du choix du processeur-jeton : l'*initiateur* n'est pas forcément élu processeur-jeton. Un processeur ne peut être candidat que s'il a reçu correctement tous les messages précédemment acquittés. Autrement dit, un processeur ne sera élu processeur-jeton que s'il n'a perdu aucun message estampillé (confirmé par la dernière valeur d'estampille qu'il a reçue). Dans le cas où il existe plusieurs processeurs candidats, il faut effectuer un choix arbitraire : par exemple élire le processeur ayant le plus petit numéro sur l'anneau .

4.3.1.1.3. A propos des 5 propriétés

4.3.1.1.3.1. Diffusions spontanées

Ce protocole de diffusion permet aux émetteurs de diffuser spontanément leur message à travers le groupe, sans les ordonner.

4.3.1.1.3.2. Diffusions concurrentes

Les processeurs peuvent diffuser des messages à tout moment, et ceci sans les ordonner. C'est le processeur-jeton qui met en oeuvre l'ordonnancement des messages, en leur attribuant un ordre unique en les estampillant. Remarquons que contrairement à [KTH90], le processeur-jeton ne diffuse que l'ordre dans lequel les messages doivent être remis et non les messages eux-mêmes. Par conséquent, ce protocole satisfait la propriété de diffusion concurrentes.

4.3.1.1.3.3. Nombre minimal d'annulation de messages

Pour valider un message, il faut nécessairement que ce dernier ait été reçu par L sites-jeton.

4.3.1.1.3.4. Pas d'exclusion abusive

Un émetteur peut exclure abusivement un processeur dont il n'a pas reçu l'acquittement, sous réserve que cet émetteur soit capable de communiquer avec une majorité.

4.3.1.1.3.5. Pas de contamination du système

Si on veut garantir la propriété d'ordre total en présence de défaillances, il faut que L soit supérieur ou égal à la majorité de processeurs sinon un message peut être perdu : aucun processeur appartenant au nouvel anneau ne l'ayant reçu.

En effet, considérons la situation suivante où parmi les L processeurs (avec L inférieur ou égal à la majorité) ayant validé le message m, $(L-P)$ sont défaillants et les P autres processeurs ne communiquent pas avec le nouveau processeur-jeton, aucun processeur parmi les $N-L$ n'a reçu le message m. Ce message est perdu, alors qu'il a été validé par L processeurs.

4.3.1.1.4. Conclusion

Le temps de cycle du jeton et le choix de L déterminent la latence et la résilience du protocole. Si le trafic est faible, le jeton ne circule pas, ce qui diffère d'autant la validation du message. Le choix de L est un compromis entre résilience et variance du temps de réponse.

Enfin si l'on veut garantir la non perte de messages validés en dépit de défaillances, L doit être supérieur ou égal à la majorité. Plus L est grand, plus le coût de stockage est élevé. Si L est faible, en un cycle le jeton valide de nombreux messages. Par contre le nombre de processeurs nécessaire à la validation d'un message est faible. Par voie de conséquence le risque de perte d'un message validé est accru.

4.3.1.2. Protocole 4 : A fault-tolerant protocol for atomic broadcast

V. Gligor, W. Luan, "A fault-tolerant protocol for atomic broadcast" [GL90].

4.3.1.2.1. Modèle

4.3.1.2.1.1. Modèle d'architecture

Ce protocole permet à un ensemble de processeurs de communication d'échanger des messages sur un médium de communication point-à-point. Le temps de transmission d'un message est supposé borné. Il n'existe pas d'horloge globale dans le système gérant le déroulement du protocole, mais uniquement une horloge locale à chaque processeur.

4.3.1.2.1.2. Mode de défaillance

- **mode de défaillance des processeurs** : la défaillance d'un processeur est de type **arrêt sur défaillance** : les fautes d'omission ne sont pas tolérées.
- **mode de défaillance du médium de communication** : le médium de communication est sujet aux **fautes d'omissions** avec un partionnement du système possible. Pour cette raison deux hypothèses sont ajoutées pour garantir la terminaison du protocole (cf paragraphe 4.3.1.2.2.).

4.3.1.2.1.3. Propriétés caractéristiques

Le protocole garantit les propriétés de **Consensus unanime et d'Ordre total.** Pour garantir la propriété de **Terminaison**, deux hypothèses sont nécessaires :

- Hypothèse 1 : **"Eventual Message-Delivery"** : si un processeur i envoie un message à un processeur j (d'une façon répétitive) et si i attend une réponse de j alors i recevra à la longue la réponse de j,
- Hypothèse 2 : **"Eventual Process-Recovery"** : un processeur défaillant revient à la longue dans un état correct.

4.3.1.2.2. Description du protocole

4.3.1.2.2.1. Principe

Ce protocole permet de fixer un ordre total sur un ensemble de messages diffusés à un groupe de N processeurs. Les messages sont diffusés spontanément, et reçus chez les processeurs dans un ordre quelconque. Sur réception d'un message, un processeur l'insère

directement dans une file d'attente de messages non validés et arme un réveil de validation. Sur débordement de sa file d'attente ou sur déclenchement de son réveil, un processeur, émetteur et/ou récepteur, invoque le protocole de validation.

Ce protocole, dans un environnement sans faute, est composé de trois phases qui sont successsivement, la phase d'invitation, la phase de notification et la phase de validation. L'initiateur de ce protocole est appelé l'invitant et les processeurs ayant accepté l'invitation, les invités. Les deux premières phases permettent d'élire un invitant unique et un groupe majoritaire d'invités, et la dernière phase permet de valider l'ordre de remise imposé par l'invitant.

Au cours de la **phase d'invitation**, l'initiateur (invitant) diffuse la liste contenant la séquence ordonnée de tous les messages qu'il a déjà validés au cours des précédentes invocations du protocole (cette liste est appelée dans la suite $COML_{init}$). Sur réception de cette liste, un processeur j devient invité et vérifie s'il est plus en avance que l'invitant ($COML_j > COML_{init}$). Si oui, (l'initiateur a manqué au moins une validation), j refuse cette invitation et diffuse la liste des messages qu'il a validés. Sinon, j accepte l'invitation (s'il lui manque des messages validés, il les redemande à l'initiateur). Si j était lui-même un invitant, j est perdant et il donne ses voix à i en lui fournissant la liste des réponses qu'il a reçues au cours de sa propre exécution. Dorénavant toutes les réponses qui étaient destinées à j, seront envoyées à i.

Si un initiateur i, lors de la réception des réponses, reçoit un message d'invitation d'un autre processeur j, ceci peut conduire à un phénomène de contention entre i et j s'ils ont validé la même séquence de messages ($COML_j = COML_j$). Le protocole fait donc appel à un mécanisme de priorité afin d'élire un seul invitant parmi tous les invitants potentiels. L'assignation initiale des priorités à chaque processeur n'est pas décrite dans cet article. Si la priorité de j est plus petite, i ignore l'invitation de j et continue sa propre exécution du protocole. Sinon i est perdant et accepte l'invitation de j.

Au cours de la **phase de notification**, un invitant i annule son invitation s'il a reçu au moins un refus ou s'il n'a pas reçu au moins $[N/2 + k]$ réponses positives (les voix qui lui ont été données dans la phase 1 par un invitant perdant sont prises en compte dans ce calcul). Le rôle de k permet d'avoir une marge de sécurité pour les défaillances au cours des phases 2 et 3 (en deçà de k défaillances, les messages seront quand même validés).

Si i obtient plus de $[N/2 + k]$ réponses positives, il détermine deux listes : une première contenant le plus grand ensemble de messages non validés reçus par les invités (cette liste est triée selon un ordre arbitraire fixé par i), et une seconde (appelée *COHLinit*) contenant les identificateurs des $[N/2 + k + 1]$ invités. i diffuse ces deux listes aux membres de *COHLinit*, puis entre dans la dernière phase de son protocole.

La **phase de validation** se déroule comme suit. Dès que i reçoit une majorité d'acquittements des membres de *COHLinit*, il diffuse la validation à tous les membres du groupe (invités ou non), et remet la séquence ordonnée de messages à son hôte. Dans cette phase, l'invitant doit obligatoirement diffuser le contenu des messages et non pas seulement leur identificateur à cause des processeurs non invités : en effet ces derniers n'ont pas participé aux deux premières phases et donc n'ont pas nécessairement reçu ces messages. Si i reçoit un message d'annulation provenant d'un membre j de *COHLinit* (suite à une omission du réseau, j n'a pas reçu les deux listes et donc a annulé sa participation à l'exécution du protocole), i exclut j de *COHLinit*. Si *COHLinit* devient inférieur à $(N/2+1)$, i diffuse l'annulation de son invitation à tous les processeurs.

4.3.1.2.2.2. Défaillance de l'invitant

En fonctionnement normal, un processeur invité détecte la défaillance de l'invitant sur non réception de la décision finale. Afin de satisfaire la propriété de terminaison cet invité prend l'initiative de la décision finale. Connaissant l'identité de tous les membres de *COHLinit*, il essaie d'obtenir un consensus sur la décision finale. Il invoque le protocole de terminaison décrit ci-dessous.

4.3.1.2.2.3. Défaillance d'un invité

Le protocole ne possède pas de mécanisme de détection de défaillance d'un processeur (différent d'un invitant). L'invitant continue son protocole de validation tant qu'il conserve une majorité d'invités. Si tel n'est pas le cas, il annule son invitation et en attend une autre.

4.3.1.2.2.4. Protocole de terminaison

L'initiateur de ce protocole, appelé *surrogate*, est un processeur invité ayant détecté la défaillance du processeur invitant au cours de la phase de validation. Son but est de parvenir à une des deux conditions suivantes afin de satisfaire la propriété de terminaison du protocole de diffusion.

Les conditions sous lesquelles, un processeur invité annule un message sont :

- il sait qu'une majorité d'invités a annulé sa participation au cours de la phase de notification du protocole de validation,
- ou il a reçu une annulation d'un autre invité.

Les conditions sous lesquelles, un processeur invité valide un message sont :

- il a obtenu une majorité de réponses favorables des invités,
- ou il a reçu une validation d'un autre invité.

Le *surrogate* diffuse un message de recouvrement à tous les invités tant qu'une des deux conditions précédentes n'est pas atteinte. Un invité, *j*, sur réception d'un tel message, renvoie au *surrogate* la dernière décision qu'il a prise ou qu'il a reçue au sujet de ces messages pendants. Si *j* a reçu la décision de l'invitant, il la renvoie au *surrogate*, cette réponse pouvant être soit une validation soit une annulation (l'invitant n'avait pas obtenu une majorité de réponses donc avait dû annuler son invitation). S'il n'a pas reçu la décision finale de l'invitant, il envoie son acquittement. Sur réception de ces réponses, le *surrogate* diffuse à tout le groupe une validation s'il a reçu au moins une validation ou a obtenu une majorité d'acquittements ; sinon il diffuse une annulation si la condition d'annulation est satisfaite.

4.3.1.2.3. A propos des cinq propriétés

4.3.1.2.3.1. Diffusion spontanée

Un processeur est autorisé à diffuser son message à tout instant, même si une exécution du protocole de validation est en cours.

4.3.1.2.3.2. Diffusions concurrentes

Plusieurs processeurs peuvent simultanément diffuser leur message. C'est le processeur initiateur du protocole en trois phases qui ordonne les messages en attente de validation. Ce protocole permet de gérer les diffusions concurrentes.

4.3.1.2.3.3. Nombre minimal d'annulation de messages

Dans ce protocole, l'aspect succès d'une diffusion est préférée à l'aspect terminaison. Tant qu'il existe un processeur non défaillant ayant en attente un message non validé, ce processeur est susceptible d'invoquer le protocole de validation. Le protocole de validation aboutit à la validation tant qu'il existe une majorité de processeurs communiquant avec l'initiateur du protocole.

4.3.1.2.3.4. Pas d'exclusions abusives

Il n'existe pas de procédure d'exclusion de processeurs. La liste des membres du groupe ayant permis la validation est construite dynamiquement à chaque invocation du protocole.Tout processeur étant susceptible d'être un initiateur, le risque de blocage sur défaillance de ce dernier est limité : le système se bloque dès qu'une majorité de processeurs ont défailli.

4.3.1.2.3.5. Pas de contamination du système

Le premier type de contamination à éviter est dû au comportement d'un processeur incorrect. Lorsqu'un processeur décide de valider un message, il doit nécessairement obtenir l'accord d'une majorité de processeurs. De plus, un processeur ne remet un message à son hôte que s'il a auparavant remis le message précédent. Ceci permet donc d'éviter le premier risque de contamination.

Le second type est provoqué par une violation des hypothèses de bornes. En cours de fonctionnement, l'invitant vérifie qu'il appartient à la partition majoritaire du système, et un invité vérifie qu'il communique avec un invitant. Un processeur appartenant à la partition minoritaire est uniquement autorisé à remettre les messages validés à son hôte, mais n'est en aucun cas susceptible de valider lui-même un message.

Ces aspects du protocole permettent donc de satisfaire la propriété de non contamination du système.

4.3.1.2.4. Conclusion

Ce protocole est basé sur le partage du coût de la validation entre plusieurs messages, mais le temps de réponse d'une diffusion dépend de l'importance du trafic du système, ainsi que des possibilité de communication dans le système. Le temps de réponse n'est donc pas fixe et peut varier dans des proportions importantes : un nombre minimum de messages doivent être reçus chez un processeur pour que ce dernier invoque le protocole en trois phases décrit précédemment.

Ce protocole n'exclut pas les processeurs défaillants. Lors de chaque validation, une liste des processeurs ayant permis la validation est dynamiquement élaborée.

Un inconvénient de ce protocole concerne le coût en stockage des messages qui ont été validés par une majorité mais pas par tous les processeurs du groupe. En effet, les processeurs ayant validé ces messages doivent être capable de les retransmettre sur simple demande.

4.3.1.3. Protocole 5 : Atomic broadcast for redundant broadcast channels

F. Cristian, "Atomic broadcast for redundant broadcast channels" [CAS85].

4.3.1.3.1. Modèle

4.3.1.3.1.1. Architecture du système

Le protocole de diffusion atomique "Lazy Forwarding" appartient à la classe des protocols dits synchrones. Ces protocoles garantissent l'atomicité des diffusions non pas en faisant intervenir des messages d'acquittement comme [BJ87], [VRB89], [GL90] et [MA91] mais en s'appuyant sur le temps écoulé. Chaque processeur a accès à une horloge locale fiable synchronisée dont le décalage maximum par rapport à l'heure réelle est une constante ρ, et telle que l'écart entre deux horloges est inférieur ou égal à une constante ε. Le médium de communication permet la diffusion et est appelé canal de diffusion. Le degré de redondance du médium est égal $f+1$, f désignant le nombre de fautes tolérées. Chaque canal est identifié par un numéro. Lorsqu'un processeur reçoit un message, il connaît le numéro du canal sur lequel la réception a eu lieu.

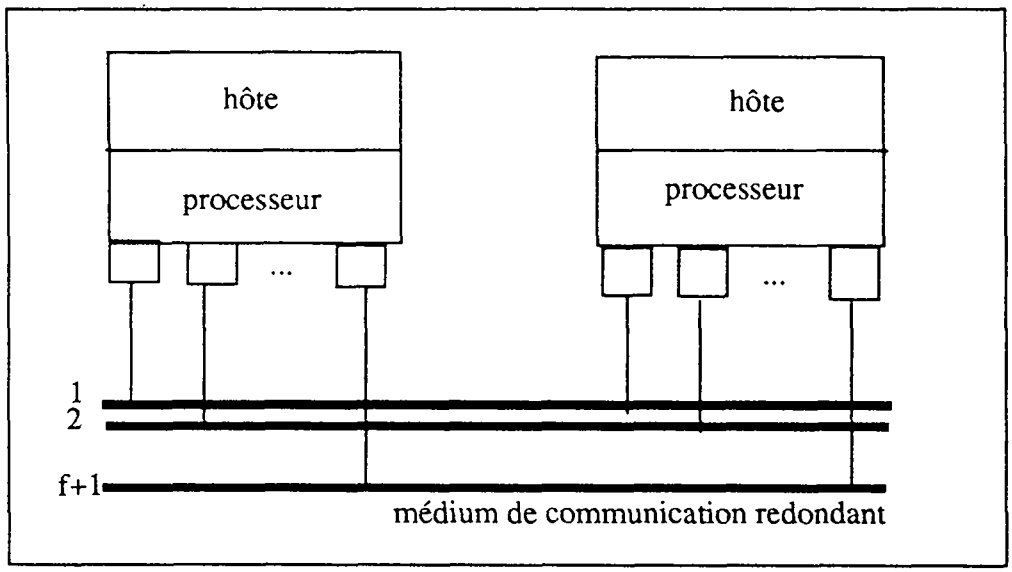

figure 10. architure du système

4.3.1.3.1.2. Mode de défaillance

- **Mode de défaillance des processeurs** est à arrêt sur défaillance ou fail-stop.
- **Mode de défaillance du médium de communication** : le bus à diffusion est sujet aux omissions (perte, duplication de messages), mais de par son degré de redondance la probabilité de partionnement du système est rendue négligeable.

4.3.1.3.1.3. Propriétés caractéristiques

Le protocole de diffusion atomique garantit les trois propriétés de **consensus unanime**, d'**ordre total** et de **terminaison**. Chaque message envoyé par un processeur à l'heure T est délivré à l'heure $T+\Delta$ sur l'horloge locale de chaque récepteur (cf paragraphe 4.3.1.3.2.1. pour le calcul de Δ).

4.3.1.3.2. Description du protocole

4.3.1.3.2.1. Principe

Lazy Forwarding n'est pas basé sur le principe "two phase-commit" comme [BJ87], [VRB89], [GL90] et [MA91], mais utilise le principe "one phase-dissemination" et se base sur le temps écoulé pour valider.

Un message diffusé à l'heure T sur l'horloge de l'émetteur est soit délivré par tous les corrects à l'heure $T+(k+1)(\delta+\varepsilon)$ soit par aucun d'entre eux, avec $f=2k+1$ le nombre de fautes tolérées, δ le temps de transmission d'un message d'un processeur à un autre et ε l'écart maximal pouvant exister entre deux horloges. La constante $(k+1)(\delta+\varepsilon)$ notée Δ est appelée délai de terminaison du protocole, et le temps $T+\Delta$, appelé échéance de remise représente l'heure à laquelle les messages d'estampille T doivent être remis à l'hôte.

L'émetteur diffuse son message m sur les $f+1$ canaux, dans l'ordre croissant de leur numéro (les canaux sont numérotés de 1 à $f+1$), et $f+1$ représente le nombre minimal de messages qu'il est nécessaire d'envoyer afin de garantir les propriétés d'unanimité et de terminaison d'une diffusion atomique. Quand un processeur p reçoit un tel message à l'heure d'arrivée U (heure délivrée par l'horloge locale de p), il accepte ce message m si U ne dépasse pas $T+h(\delta+\varepsilon)$, avec h le nombre de fois où m a été retransmis. Soit c le plus grand canal sur lequel p a reçu m : si $c<f+1-h$ et $h<=k$, alors p diffuse m sur les bus $c+1,...f+1-h$ (dans cet ordre). Que p rediffuse ou non une copie de m, il conserve le message dans un buffer jusqu'à ce que l'échéance de m soit atteinte, et le remet alors à l'hôte.

4.3.1.3.2.2. Défaillance d'un processeur

Dans ce protocole, aucun mécanisme de détection de défaillance de processeurs en cours de fonctionnement du système n'est implémenté. Ainsi un processeur défaillant et un processeur qui reçoit un message ne nécessitant pas d'être rediffusé ($c>f$), sont perçus de la même manière, par les autres processeurs. En conséquence, le silence d'un processeur ne peut pas être interprété comme signe de sa défaillance.

4.3.1.3.3. A propos des cinq propriétés

4.3.1.3.3.1. Diffusion spontanée

Contrairement au protocole de [GT89] (également synchrone), un processeur est autorisé à diffuser un message à tout instant. Ce protocole gère les diffusions spontanées.

4.3.1.3.3.2. Diffusions concurrentes

Les diffusions concurrentes sont gérées par le protocole. Deux messages portant la même heure d'émission sont différenciés par l'idendificateur de la source.

4.3.1.3.3.3. Nombre minimal d'annulation de messages

Dès qu'un message a été diffusé par l'émetteur, disons au temps T, ce message est obligatoirement validé au temps $T+\Delta$ (heure de remise de tous les messages diffusés au temps T), et ceci même en présence de f défaillances. La propriété de minimalité du nombre d'annulation est donc satisfaite.

Pour qu'un message soit validé par un processeur, il suffit que ce message soit reçu avant l'heure de remise. Si le nombre de fautes maximum est dépassé, l'accord unanime n'est plus garanti. Certains processeurs annulant et d'autres validant le même message.

4.3.1.3.3.4. Pas d'exclusion abusive

Il n'y a pas de procédure d'exclusion dans ce protocole, donc la propriété d'exclusion abusive est satisfaite.

4.3.1.3.3.5. Non contamination du système

Le facteur susceptible de contaminer le système concerne la violation des bornes supposées par le protocole. Tout d'abord, le silence d'un processeur ne pouvant être interprété comme signe de sa défaillance, le protocole est incapable de détecter un dépassement du nombre maximum de fautes tolérées, et il peut donc conduire à un comportement incontrôlé du système.

Ensuite, est-il possible que malgré le protocole probabiliste de synchronisation des horloges, l'écart maximum entre deux horloges reste supérieur à ε ?

4.3.1.3.4. Conclusion

Ce protocole est basé sur une attente qui fait intervenir le délai maximum ; en d'autres termes pour être sûr que l'on peut délivrer un message il faut toujours se placer dans le pire cas. Par contre en présence de défaillances, la borne Δ permet alors de conclure en un temps constant, alors que les autres protocoles doivent effectuer une procédure de recouvrement. Un autre attrait de ce protocole est la faible complexité en nombre de messages : en l'absence de défaillance, elle est égale à $f+1$, ce nombre étant indépendant du nombre de processeurs appartenant au groupe de diffusion. En cas de défaillances, cette complexité est égale à $n(f-1)+2$ dans le pire cas.

La non-détection du dépassement de f peut poser problème.

- Si ce protocole est utilisé avec un algorithme déterministe de synchronisation d'horloges (on a alors ε voisin de δ-d avec d=délai de transmission minimum) et si l'on veut tolérer f fautes pour $2f+1$ processeurs, on a alors $\Delta=(k+1)(2\delta-d)=(\frac{f+1}{2})(2\delta-d)$.

- Si de plus, on suppose $\delta = 2d$ on a alors $\delta \sim \frac{3}{2}(f+1)d$

- et si $\delta = 3d$ on a $\delta \sim \frac{5}{2}(f+1)d$.

On peut comparer ce temps de réponse avec celui obtenu par un protocole en deux phases comme ABCAST ou AMp où l'on a : temps de réponse = $2(f+1)d$ avec d = délai de transmission minimum.

4.3.1.4. Protocole 6 : Reliable Broadcast protocols

K. Birman, T. Joseph, "Reliable communication in the presence of failures" [BJ87].

4.3.1.4.1. Modèle

4.3.1.4.1.1. Architecture du système

Les processeurs sont connectés à un canal de communication. Les délais de transmission sont supposés bornés. Chaque processeur a accès à une horloge locale. Les horloges locales ne sont pas synchronisées.

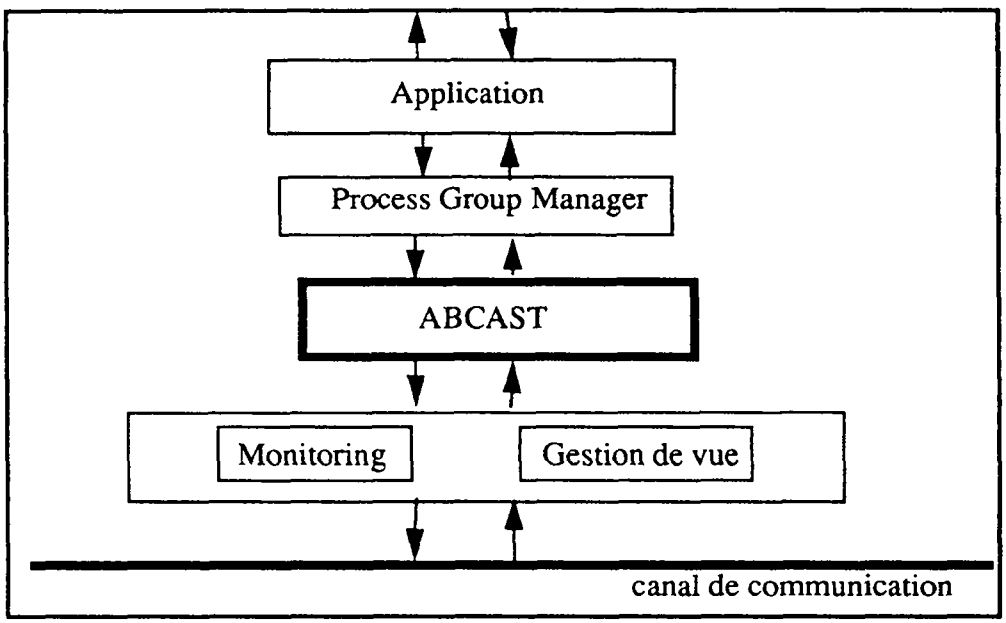

figure 11. architecture du système

4.3.1.4.1.2. Mode de défaillance

Le mode de défaillance des processeurs est de type **arrêt sur défaillance** : les fautes d'omission ne sont pas tolérées. Seul le médium de communication est sujet aux **fautes d'omission** (les messages peuvent être perdus, doublés, reçus dans le désordre ...).

4.3.1.4.1.3. Propriétés caractéristiques

Le protocole garantit les trois propriétés caractéristiques d'une diffusion atomique à savoir l'**Unanimité**, l'**Ordre total** et la **Terminaison**.

4.3.1.4.2. Description du protocole

4.3.1.4.2.1. Principe

ABCAST est un protocole de diffusion fiable se déroulant en deux phases : dans la première phase, un émetteur diffuse son message à travers le groupe. Chaque récepteur calcule l'estampille qu'il propose pour le message reçu : il incrémente un compteur local et concatène cette valeur à son identificateur. Il marque comme indélivrable le message, et l'acquitte en renvoyant son estampille. L'émetteur collecte tous les acquittements puis diffuse la validation de son message en lui assignant son estampille finale, c'est-à-dire la plus grande estampille

proposée. Sur réception de cette validation, les récepteurs affectent l'estampille finale au message et le marquent délivrable. Un message délivrable est remis à l'application dans l'ordre croissant des estampilles finales.

De plus, les processeurs ont à leur disposition deux services : un **service de monitoring des processeurs** et un **service de gestion de vue**. Le service de monitoring permet de détecter la défaillance d'un processeur : périodiquement, chaque processeur diffuse un message de vie à travers le groupe : dès qu'un processeur détecte qu'il n'a pas reçu le message de vie d'un processeur P, il le considère comme défaillant et fait appel au service de gestion de vue pour que ce dernier mette à jour la vue du groupe en excluant P.

4.3.1.4.2.2. Détection de la défaillance d'un récepteur

Si la défaillance d'un récepteur n'a pas été détectée par le service de monitoring (période d'invocation trop grande), cette défaillance est alors détectée par un émetteur à la fin de sa phase de dissémination : sur non-réception d'un acquittement, l'émetteur valide son message en ignorant les processeurs silencieux. Il invoque le service de gestion de vue pour exclure ces processeurs de la vue courante.

4.3.1.4.2.3. Détection de la défaillance d'un émetteur

Sur détection de la défaillance d'un émetteur, un récepteur ayant localement un message non validé termine la diffusion pendante en devenant le "coordinateur" du groupe. Il interroge les autres processeurs du groupe pour connaître l'état du message : jamais reçu, reçu ou validé. Si aucun processeur n'a reçu la validation du message, le coordinateur réexécute entièrement ABCAST, sinon il rediffuse uniquement la validation du message, avec l'estampille finale que l'ancien émetteur avait proposée.

4.3.1.4.2.4. Algorithme de gestion de vue

Cet algorithme ne peut-être exécuté que par un seul processeur : le "view manager". Il s'agit du processeur appartenant au groupe depuis le plus longtemps, et référencé comme tel dans la vue courante. Cet algorithme se déroule en deux phases, respectant les propriétés d'une diffusion atomique.

Dans la première phase, le view manager diffuse une extension de la vue courante en excluant le processeur défaillant. Sur réception de cette extension, chaque récepteur renvoie un acquittement positif au processeur view manager s'il s'agit de la première extension qu'il reçoit ou si la précédente qu'il a reçue est moins à-jour que celle-ci. Dans le cas contraire il acquitte négativement et rejette cette extension.

Le view manager collecte les acquittements :

- soit toutes les réponses sont positives : le view manager diffuse la nouvelle vue à travers le groupe (deuxième phase) ce qui termine son algorithme,
- soit il lui manque des acquittements : il considère que les silencieux sont défaillants et recommence donc la phase 1 de son algorithme avec une nouvelle extension de vue excluant les silencieux,
- soit il reçoit un acquittement négatif, il recalcule une nouvelle extension de vue prenant en compte celle reçue dans l'acquittement négatif et reprend la phase 1 de son algorithme.

4.3.1.4.2.5. Détection de la défaillance du processeur view manager

Lorsqu'un processeur détecte la défaillance du view manager (via le "monitoring mechanism") il fait appel au successeur du view manager, qui devient ipso-facto le nouveau view manager.

Le passage automatique d'un view manager à son successeur pose un problème de cohérence au niveau du groupe : quel est le view manager courant ? En effet que se passe-t-il dans le cas où le view manager courant *P1* et son successeur *P2* ne peuvent pas communiquer entre eux ? *P1*et *P2* exécutent parallèlement l'algorithme de gestion de vue, *P1* excluant *P2* de la vue et *P2* excluant *P1*de la vue. Sur réception de ces deux mises-à-jour contradictoires, comment doivent se comporter les récepteurs ? Qui doivent-ils acquitter positivement, *P1* ou *P2* ? Cela n'entraîne-t-il pas des exclusions en cascade ?

4.3.1.4.3. A propos des 5 propriétés

4.3.1.4.3.1. Diffusions spontanées

Dans ABCAST, les émetteurs initialisent une diffusion dès qu'ils ont un message à émettre, et ceci sans devoir attendre un top d'horloge. La propriété de diffusion spontanée est donc satisfaite.

4.3.1.4.3.2. Diffusions concurrentes

Des messages émis simultanément à travers le groupe sont estampillés, d'une façon unique sur réception des acquittements de tous les membres du groupe, ce qui permet de gérer les diffusions concurrentes tout en assurant la propriété d'ordre total.

4.3.1.4.3.3. Nombre minimal d'annulation de messages

Dès qu'un message est diffusé il est validé soit par l'émetteur lui-même, soit par le nouveau coordinateur en cas de défaillance de l'émetteur. Ceci permet donc à ABCAST de satisfaire la propriété du nombre minimal d'annulation de messages.

4.3.1.4.3.4. Pas d'exclusion abusive

Dès qu'un processeur détecte qu'un membre du groupe est défaillant, il appelle le view manager qui directement l'exclut du groupe, sans aucun test préalable. Aucun consensus majoritaire n'est requis. Cette technique peut entraîner par conséquent des exclusions abusives. Lorsqu'un processeur reçoit un message l'informant de son exclusion du groupe, il se suicide.

4.3.1.4.3.5. Non contamination du système

Dans ABCAST, la contamination du système par un émetteur incorrect est possible : en effet, même si un émetteur ne communique pas avec une majorité de processeurs il est autorisé à valider son message. Deux partitions concurrentes sont alors en présence dans le système ce qui entraine obligatoirement une incohérence dans l'ordre de remise des messages et donc une violation de l'ordre total, propriété caractéristique d'une diffusion atomique.

Une autre contamination du système est rendue possible lors de la défaillance simultanée de l'émetteur et d'un récepteur. En effet supposons que l'émetteur défaillant ait diffusé la validation de son message avec l'estampille finale proposée par *Pi*, que seul *Pi* ait reçu cette validation, et que *Pi* défaille lui aussi après avoir remis le message. L'ordre trouvé par le

nouveau coordinateur sera différent de l'ordre attribué initialement. La propriété d'ordre total n'est plus assurée, et l'historique de *Pi* n'est pas un préfixe de l'historique d'un processeur correct.

4.3.1.4.4. Conclusion

Dans ABCAST, un processeur ne contrôle pas son appartenance à la partition majoritaire pour valider son message. Si un partitionnement survient, les partitions peuvent évoluer indépendamment. N'importe quel processeur peut décider d'exclure un processeur silencieux. Ceci peut donner lieu à des exclusions abusives.

Cependant, on notera que ABCAST est un des premiers protocoles de diffusion atomique à gérer les diffusions concurrentes selon une approche décentralisée.

4.3.1.5. Protocole 7 : Fast ABCAST protocol

K. Birman, A. Schiper, P. Stephenson, "Fast Causal Multicast" [BSS90].

Ce protocole de diffusion atomique, permet de gérer l'existence de plusieurs groupes de diffusion, en privilégiant l'aspect performance à celui de fiabilité. Il suppose l'existence d'un système de communication fiable, une seule défaillance de processeur par groupe de diffusion et les messages émis par un processeur sont reçus dans leur ordre d'émission), et ne tolère qu'une seule défaillance de processeur par groupe de diffusion. Il est implémenté dans ISIS [BCJ90].

4.3.1.5.1. Modèle

4.3.1.5.1.1. Modèle d'architecture

Les processeurs sont connectés à un canal de communication. Les délais de transmission sont supposés bornés. Chaque processeur a accès à une horloge locale. Les horloges locales ne sont pas synchronisées. Le prrotocole Fast ABCAST est implémenté au dessus d'un protocole de communication fiable (protocole semblable à TCP (Transport Control Protocol) [POS80], garantissant la non-perte, la non-duplication et le séquencement des messages échangés entre deux processeurs donnés.

4.3.1.5.1.2. Mode de défaillance

Le mode de défaillance des processeurs est de type **arrêt sur défaillance** : les fautes d'omission ne sont pas tolérées. Le système de communication est supposé **fiable**, et de plus si un processeur *P* émet le message *m1* puis le message *m2* alors le processeur *Q* reçoit d'abord le message *m1* puis le message *m2*.

4.3.1.5.1.3. Propriétés caractéristiques

Le protocole garantit les trois propriétés caractéristiques d'une diffusion atomique à savoir l'**Unanimité**, l'**Ordre total** et la **Terminaison**.

4.3.1.5.2. Description du protocole

4.3.1.5.2.1. Principe

Contrairement aux protocoles de diffusion atomique étudiés dans ce chapitre, ce protocole de diffusion atomique est basé sur un protocole de diffusion causale. En effet toute diffusion atomique utilise sur la primitive de diffusion causale Fast CBCAST. Nous faisons précéder la description de Fast ABCAST par un rappel de Fast CBCAST. Fast CBCAST utilise deux mécanismes d'estampillage : un temps logique [LAM78], noté LT et un vecteur temps, noté VT.

4.3.1.5.2.1.1. Temps logique LT

Chaque processeur maintient localement un compteur appelé temps logique et noté *LT*. Dans ce protocole un processeur *Pi* retarde la remise d'un message *m* d'estampille *LT(m)* jusqu'à ce qu'il reçoive, de chaque processeur du groupe, un message *m'* d'estampille *LT(m')*, avec *LT(m')>=LT(m)*. Ceci, bien évidemment ne marche que si tout processeur *Pj* du groupe diffuse un flot infini de messages. En raison du caractère assez peu réaliste de cette hypothèse, il est nécessaire d'introduire un mécanisme particulier, appelé *flush du canal de communication* : si le canal de communication de *Pj* à *Pi* est *flushé* à l'heure *LT(m)* alors *Pi* ne recevra jamais un message *m'* émis par *Pj*, avec *LT(m') <LT(m)*.

Plus formellement, ce protocole est le suivant :

- soit *Pi* un processeur appartenant au groupe de diffusion *g*.
  - pour émettre un message *m* dans le groupe *g*, *Pi* procède comme suit :
    - *a1) LT(pi) = LT(Pi) + 1 ;*
    - *a2) LT(m) = LT(Pi)* (le message *m* est estampillé avec *LT(Pi)*) ;
    - *a3) send(m, LT(m)).*
  - sur réception de *send(m, LT(m))* émis par *Pi*, *Pj* procède comme suit :
    - *b1) LT(Pj) = max(LT(Pj), LT(m)).*
    - *b2) Pj* retarde la remise de *m* jusqu'à ce que pour tout *k<>i*, le canal de communication entre *Pi* et *Pj* soit *flushé* pour le temps *LT(m)*.
    - *b3) Pj* ne retarde pas la remise du message dont il est l'émetteur.

Le *flushing* du canal de communication entre *Pj* et *Pi* peut être résolu grâce à des messages de vie. Par exemple :

- *Pi* envoie à *Pj* un message de vie dont l'estampille *LT(Pi)*, n'a pas été préalablement incrémentée (contrairement au protocole de base *LT).*
- sur réception de ce message de vie, *Pj* met-à-jour *LT(Pj)* avec *LT(Pj) =max(LT(Pj), LT(msg. vie))*, puis acquitte *Pi* (l'acquittement est estampillé avec *LT(Pj)*).
- sur réception de cet acquittement, *Pi* met-à-jour *LT(Pi)*, avec *LT(Pi) =max(LT(Pj), LT(acq)* (comme c'est le cas dans le protocole de base *LT*).

Ainsi, si aucun nouveau message n'est en cours de diffusion, alors le *flushing* permet de forcer *LT(Pi)* et *LT(Pj)* à la même valeur.

Remarquons qu'un récepteur *Pj* ne remet un message *m* à son hôte que s'il a reçu tous les messages d'estampille inférieure ou égale à *m* (voir étape *b2* du protocole). Par contre, l'émetteur *Pi* remet immédiatement *m* à son hôte (voir étape *b3* du protocole). En conséquence, un processeur qui n'a été que récepteur a remis les messages selon l'ordre imposé par *LT*, contrairement à un processeur qui a été émetteur.

L'examen de ce protocole montre que l'ordre causal tel qu'il a été défini au paragraphe 4.1. est approché du fait que *LT* est incrémenté sur réception d'un message et non lors de sa remise.

L'inconvénient de ce protocole est le mécanisme de *flush du canal*, invoqué par chaque récepteur *Pj* sur réception d'un message *m*. Ce mécanisme est responsable d'un coût élevé en nombre de messages en effet, $n^2 - 3n + 3$ messages sont nécessaires pour qu'en l'absence de diffusion concurrentes et en l'absence de défaillance, un message *m* soit remis par tous les processeurs du groupe de diffusion (complexité du protocole en $O(n^2)$).

4.3.1.5.2.1.2. Vecteur Temps *VT*

Chaque processeur maintient localement un vecteur de compteurs appelé vecteur temps et noté *VT*. La solution repose sur le même principe que le mécanisme précédent, mais est moins coûteuse en nombre de messages. La composante *k* de l'estampille *VT(m)* du message *m*, notée *VT(m)[k]*, indique quels sont les messages de *Pk* remis par l'émetteur de *m* avant la diffusion de *m*. Le protocole initialement proposé par [MAR84], puis repris par [FID88], [MAT89], [LL86], [SCH88], est le suivant :

- pour diffuser un message *m* dans le groupe *g*, *Pi* procède comme suit :
  - *1) VT(Pi)[i] = VT(Pi)[i] + 1 ;*
  - *2) VT(m)[i] = VT(Pi)[i]* (*m* est estampillé avec *VT(m)*) ;
  - 3) *send(m, VT(m)[i])*
- sur réception de *send(m, VT(m)[i])* émis par *Pi :*
  - *Pj* retarde la remise de *m* jusqu'à ce que :
    - 1) *VT(m)[i] = VT(Pj)[i] + 1*
    - 2) quelque soit *k<>i : VT(m)[k] <= VT(Pj)[k]*
  - *Pi* ne retarde pas la remise de son message
- un récepteur *Pj* sur remise de *m*, met à jour son vecteur *VT(Pj)* comme suit *:*
  - *pour tout k = 1..n, VT(pj)[k] = max(VT(pj)[k], VT(m)[k]).*

On remarque que contrairement au premier protocole, le compteur *VT* est mis-à-jour lors de la remise du message à l'hôte. Il a été montré par [MAT89] et [FID88] que les vecteurs temps *VT* représentent exactement l'ordre causal d'émission. Ceci est la raison pour laquelle ces vecteurs sont préférés aux compteurs logiques *LT*.

4.3.1.5.2.1.3. Utilisation combinée de *LT* et *VT*

En l'absence de défaillance, le vecteur de compteurs *VT* serait suffisant. Comme le protocole Fast CBCAST veut tolérer les défaillances, il utilise simultanément les deux mécanismes *LT* et *VT*. *LT* est utilisé pour estampiller la vue du groupe. *LT* n'est donc incrémenté que sur changement de vue. *VT* est utilisé comme précédemment pour estampiller le message diffusé.

Pi délivre un message *m* à son hôte si *m* respecte la règle de remise *LT* (cf paragraphe 4.2.1.2.2.2.) et la règle de remise *VT* (cf paragraphe 4.2.1.2.2.3.).

Dans le cas particulier où le changement de vue fait suite à la défaillance d'un processeur *Pi*, le **protocole de flush du groupe** est exécuté par chaque membre *Pj* de cette nouvelle vue :

- *Pj* ignore dorénavant tout message provenant de *Pi*,
- pour tout message *m* diffusé par *Pi* et dont *Pj* ignore s'il a été reçu par tous les processeurs du groupe, *Pj* diffuse une copie de *m*,
- *Pj* flush le canal de communication avec tout processeur *Pk* de la nouvelle vue,
- *Pj* rejette tout message en attente d'un message *m* de *Pi*, avec *m* reçu par aucun processeur de la nouvelle vue,
- *Pj* ne maintient plus la composante *i* (associée à *Pi*) de *VT*.

4.3.1.5.2.1.4. Extension à plusieurs groupes de diffusion

Ce protocole valable pour un seul groupe de diffusion doit être généralisé si l'on veut l'adapter à plusieurs groupes de diffusion. Les modifications sont les suivantes :

- 1) étendre l'horloge unique *VT* à une horloge multiple, c'est-à-dire une horloge reflétant l'état de chaque groupe : VT_a est l'horloge associé au groupe g_a, et $VT_a(m)\ [i]$ repésente les messages diffusés par *Pi* dans le groupe g_a remis par l'émetteur de *m* avant la diffusion de *m* dans le groupe g_a.
- 2) sur réception d'un message *m* diffusé par *Pi*, le processeur *Pj* doit retarder la remise de *m* jusqu'à :
  - 1) $VT_a(m)[i] = VT_a(Pj)[i] + 1$,
  - 2) et quelque soit *k* : (*Pk* appartient à g_a et $k<>i$) : $VT_a(m)[k] <= VT_a(Pj)[k]$,
  - 3) et quelque soit *g* un groupe de diffusion : $VTg(m) <= VTg(Pj)$

La figure 9. ci-dessous illustre l'application de ces nouvelles règles. Soient *P1*, *P2*, *P3* (respectivement *P2*, *P3* et *P4*) appartenant au groupe g_1 (respectivement au groupe g_2). *P3* ne remet pas directement le message *m2* à son hôte car il appartient à g_1 et doit, par conséquent, d'abord remettre *m1* puis *m2* . Par contre, *P4* remet aussitôt *m2* à son hôte, car il appartient uniquement à g_2 et donc ne doit remettre que des messages diffusés dans g_2. La remise de *m3* n'est pas différée chez *P2*, car il a déjà reçu *m1* et est l'émetteur de *m2*.

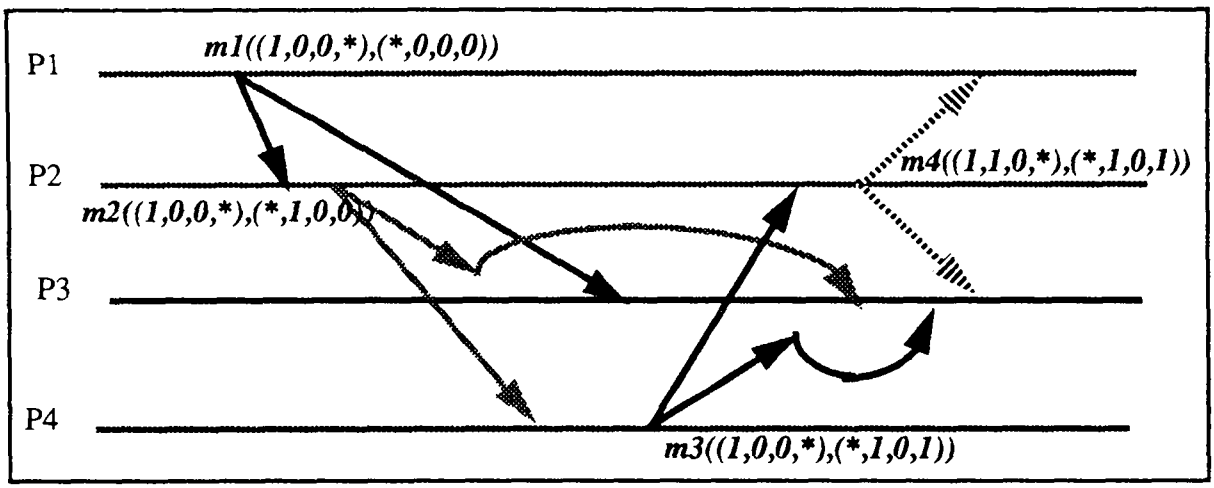

figure 12. Messages diffusés dans 2 groupes g_1 = {P1, P2, P3} et g_2 = { P2, P3, P4}

• 3) la condition d'incrémentation de *LT* et *VT* est modifiée. Avant d'énoncer cette modification, il est nécessaire de définir les deux termes suivants :

Définition 1 : groupe exclusif

Un processeur *Pi* appartient à un **groupe de diffusion exclusif** *g* si *g* a au moins deux groupes voisins. Un groupe *g* est voisin d'un groupe *g'* s'il existe un processeur *Pi* appartenant à *g* et appartenant à *g'*.

Définition 2 : processeur non-sûr

Soit *m* un message diffusé par *Pi* dans le groupe *g*. *Pi* est dit **non-sûr** dans *g* si :

• soit le dernier message reçu par *Pi* a été envoyé par un processeur *Pj* appartenant à *g'*,

• soit *g* ou *g'* est un groupe exclusif.

Condition d'incrémentation de LT et VT

Soit un processeur *Pi* appartenant au groupe de diffusion *g* :

• si *Pi* est non-sûr dans le groupe *g*, alors *Pi* incrémente à la fois l'estampille *LT* et l'estampille *VT* avant de diffuser son message *m* dans *g*.

• sinon *Pi* incrémente uniquement son estampille *VT*.

Condition de remise d'un message

Pi délivre un message *m* à son hôte si *m* respecte la règle de remise *LT* (cf paragraphe 4.2.1.2.2.2.) et la règle de remise *VT* (cf paragraphe 4.2.1.2.2.3.).

• 4) un processeur *Pi* n'est autorisé à diffuser un message au groupe *g*, que si tous les messages diffusés par *Pi* aux autres groupes, dont il est membre, ont été délivrés par au moins un processeur *Pj* différent de *Pi*. Cette condition est nécessaire pour tolérer la défaillance d'un émetteur, comme le montre la figure 10. suivante :
soit *P1* appartenant à *g1* et *P2* appartenant à *g1* et *g2*. *P1* diffuse un message *m1* dans *g1* puis défaille. Ce message *m1* n'est reçu que par *P2*, qui diffuse alors un message *m2* dans *g2*, puis défaille. Les processeurs de *g2* n'appartenant pas à *g1* remettent le message *m2* tandis que ceux appartenant à *g1*et *g2* (exemple *P3*) n'ayant pas reçu *m1*, annulent *m2*

conformément au protocole du flush du groupe : la propriété d'unanimité n'est plus respectée.

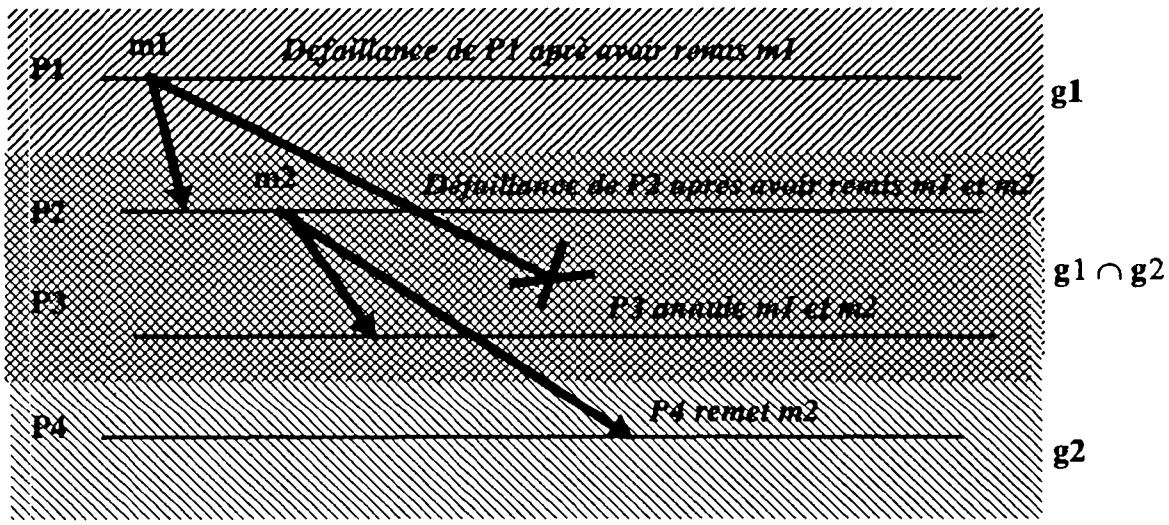

figure 13. Perte de la propriété d'Unanimité

4.3.1.5.2.1.5. Principe de Fast ABCAST

Chaque groupe de diffusion possède un processeur central, appelé processeur-jeton. Ce protocole s'exécute en deux phases : dans la première phase, l'émetteur diffuse son message dans le groupe. Dans la seconde phase, le processeur-jeton attribue l'ordre dans lequel ce message doit être remis par tous les processeurs, et diffuse cet ordre au groupe. Contrairement au protocole de Chang et Maxemchuck [CM84], il n'existe pas de phase de validation. La description formelle du protocole de diffusion atomique est la suivante :

- phase 1) un processeur *Pi* diffuse son message *m* dans le groupe de diffusion : *CBCAST(needs-order, m)*. Sur réception de ce message, le processeur-jeton attribue l'ordre de remise de *m*. Les autres processeurs mémorisent ce message et attendent, pour le remettre, d'avoir reçu l'ordre imposé par le processeur-jeton.

- phase 2) le processeur-jeton diffuse un CBCAST(*sets-order*), où *sets-order* indique l'ordre de remise du message *m*. Sur réception de ce message, les processeurs remettent *m* selon l'ordre fourni dans *sets-order*.

Ce protocole appelle les deux remarques suivantes :

Remarque 1

Le coût de ce protocole en nombre de messages dépend de deux facteurs : 1) de l'émetteur du message, 2) de la fréquence de passage du jeton. Si les diffusions tendent à provenir du même processeur *Pi* d'une façon répétitive, alors dès que *Pi* obtient le jeton, le coût est égal à un CBCAST par diffusion atomique. Par contre, si les émetteurs sont aléatoirement distribués à travers le groupe, et que le jeton reste sur le même processeur, le coût est de $1 + 1/k$ CBCAST par diffusion atomique, si l'on suppose qu'un seul message *set-order* est diffusé pour ordonner k messages.

Remarque 2

Il est possible de faire circuler le jeton entre tous les processeurs émetteurs du groupe. Aucun détail supplémentaire n'est fourni dans l'article.

Remarque 3

S'il existe plusieurs groupes de diffusion, l'ordre total n'est pas garanti à l'intersection de deux groupes (pour un exemple se référer à la figure 3. au paragraphe 2.2.).

4.3.1.5.2.2. Détection de la défaillance d'un processeur

Aucun mécanisme particulier n'est ajouté par rapport à Fast CBCAST. On rappelle que les processeurs ont à leur disposition deux services : un **service de monitoring des processeurs** et un **service de gestion de vue**. Le service de monitoring permet de détecter la défaillance d'un processeur, alors que le service de gestion de vue est chargé de la mise à jour atomique de la vue du groupe : tout processeur défaillant est exclu de la vue. Ces deux services sont implémentés à un niveau différent de Fast ABCAST. Leur principe de fonctionnement n'est pas décrit dans cet article, mais il est supposé que la défaillance d'un processeur est détectée au plus tôt, et que la vue du groupe est mise-à-jour d'une façon cohérente. Par conséquent, lors d'une défaillance, tout processeur du groupe en est directement informé, par réception d'une nouvelle vue axcluant le processeur défaillant. Sur réception de cette nouvelle vue le mécanisme de flush du groupe utilisé dans Fast CBCAST, est mis en oeuvre. Le protocole Fast ABCAST ne tolère qu'une seule défaillance de processeur par groupe de diffusion : seule, la défaillance du processeur-jeton affecte l'ordre de remise des messages. Ceci est étudié dans le paragraphe 4.3.1.5.2.3.

4.3.1.5.2.3. Détection de la défaillance du processeur-jeton

Dans le cas où la défaillance du processeur-jeton intervient au cours de sa diffusion du message CBCAST(*sets-order*), il est possible que certains processeurs du groupe aient reçu ce message alors que d'autre ne l'ont pas reçu. Sur réception de la nouvelle vue excluant le processeur-jeton, tous les processeurs de la nouvelle vue mettent en oeuvre le mécanisme de flush du groupe. Ainsi, au terme de ce flush, tous les processeurs ont la même liste de messages pendants, qu'ils ordonnent selon un algorithme déterministe, ce qui leur permet de remettre ces messages pendants à leur hôte dans le même ordre.

4.3.1.5.3. A propos des 5 propriétés

4.3.1.5.3.1. Diffusions spontanées

Ce protocole de diffusion atomique permet aux émetteurs de diffuser spontanément leur message à travers le groupe, mais ne sont pas chargé de les ordonner.

4.3.1.5.3.2. Diffusions concurrentes

Les processeurs peuvent diffuser des messages à tout moment, et ceci sans les ordonner. C'est le processeur-jeton qui met en oeuvre l'ordonnancement des messages, en leur attribuant un ordre unique en les estampillant. De façon similaire à [CM84], le processeur-jeton ne diffuse que l'ordre dans lequel les messages doivent être remis et non les messages eux-mêmes. Par conséquent, ce protocole satisfait la propriété de diffusion concurrentes.

4.3.1.5.3.3. Nombre minimal d'annulation de messages

Cette propriété est garantie du fait qu'un message est annulé ssi :

- l'émetteur du message et le processeur-jeton ne font qu'un,
- il défaille en cours de diffusion et aucun autre processeur du groupe n'a reçu ce message.

4.3.1.5.3.4. Pas d'exclusion abusive

L'exclusion des processeurs n'est pas gérée par le protocole de diffusion atomique (cf monitoring des processeurs et gestion de vue vus précédemment). Il n'est donc pas possible de savoir si cette propriété est satisfaite ou non. Cependant si ces procédures sont identiques à celles présentées dans [BJ87], alors l'exclusion abusive est tout à fait possible.

4.3.1.5.3.5. Pas de contamination du système

Dans le cas d'un partitionnement du système, plusieurs partitions peuvent évoluer parallèlement, si l'on suppose que les procédures de monitoring et de gestion de vue sont celles présentées dans [BJ87].

4.3.1.5.4. Conclusion

Ce protocole de diffusion atomique est un protocole dont les caractéristiques essentielles sont d'une part la capacité de gérér plusieurs groupes de diffusion, pouvant être d'intersection non vide, et d'autre part de privilégier l'aspect performance à celui de fiabilité. Par contre les hypothèses adoptées sont trop fortes : pas de défaillance du système de communication (nous rappelons que Fast ABCAST est implémenté au-dessus d'un protocole de communication fiable garantissant la non perte , la non duplication et le séquencement des messages échangés entre deux processeurs donnés), une seule défaillance de processeur par groupe de diffusion et les messages émis par un processeur sont reçus dans leur ordre d'émission.

4.3.2. Processeur à défaillance sur omission

4.3.2.1. Protocole 8 : AMp : A highly parallel atomic multicast protocol

P. Verissimo, P. Rodrigues, M. Baptista, "AMp : A highly parallel atomic multicast protocol" [VRB89].

4.3.2.1.1. Modèle

4.3.2.1.1.1. Architecture du système

Les processeurs de communication sont connectés à un bus à diffusion. Les délais de transmission sont supposés bornés. Chaque processeur de communication est associé à un processeur d'application ou hôte. Les processeurs supportent le protocole de communication de niveau MAC et le protocole de diffusion atomique AMp. Le système ne possède pas d'horloge globale mais chaque processeur de communication a accès à une horloge locale.

4.3.2.1.1.2. Mode de défaillance

Mode de défaillance des processeurs : les processeurs de communication sont dits à arrêt sur omissions successives (ou weak fail-silent). Après un nombre maximum *w* d'omissions successives (*w* est appelé le degré d'omission du protocole) le processeur est supposé arrêté.

Mode de défaillance du médium de communication : le bus à diffusion est sujet aux omissions (perte de messages), mais possède un degré de redondance tel que le partitionnement du système est impossible.

4.3.2.1.1.3. Propriétés caractéristiques

AMp garantit les trois propriétés caractéristiques d'une diffusion atomique : à savoir **Consensus unanime, Ordre total** et **Terminaison.**

4.3.2.1.2. Description du protocole : AMp

4.3.2.1.2.1. Principe

AMp est un protocole de validation en deux phases : la phase de dissémination d'un message et sa phase de décision.

Au cours de la première phase, l'émetteur diffuse son message estampillé à travers le groupe. Si à la fin de cette première transmission il lui manque au moins un acquittement, il recommence une nouvelle transmission. Cette première phase s'achève dès qu'une des deux conditions suivantes est remplie : 1) l'émetteur a reçu tous les acquittements à la fin d'une même transmission, 2) l'émetteur a atteint le nombre maximum de retransmissions.

Dans le cas 1) si l'émetteur reçoit au moins un acquittement de type "non-accept" **il annule son message au cours de la phase de décision, sinon il le valide.** Un message de validation n'est transmis qu'une seule fois, contrairement au message d'annulation qui doit être acquitté de tous les récepteurs.

Dans l'autre cas, l'émetteur valide son message au cours de la phase de décision. S'il lui manque un acquittement (défaillance d'un récepteur), l'émetteur fait appel au "Group Monitor" qui est chargé de gérer la vue du groupe. Sur réception d'une validation, un processeur remet le message à son hôte, s'il a déjà remis tous les messages d'estampille inférieure.

4.3.2.1.2.2. Défaillance d'un récepteur

Lorsqu'un émetteur détecte la défaillance d'un récepteur en fin de phase de dissémination ou de décision (dans le cas d'annulation), il invoque son "Group Monitor", en lui fournissant l'identificateur du processeur défaillant.

4.3.2.1.2.3. Défaillance d'un émetteur

Lorsqu'après un certain temps, un récepteur n'a pas reçu la validation d'un message, il invoque son "Group Monitor". Il lui fournit l'identificateur de l'émetteur défaillant ainsi que l'estampille du message pendant.

4.3.2.1.2.4. Algorithme déroulé par le Group Monitor

Un "Group Monitor" est un service disponible sur chaque processeur, permettant d'une part d'effectuer la mise à-jour cohérente de la vue en cas de modification du groupe et d'autre part, de terminer les diffusions pendantes.

Lorsque plusieurs Group Monitors sont invoqués simultanément (cas où la défaillance d'un émetteur est simultanément détectée par plusieurs processeurs) il est nécessaire de procéder à l'élection d'un Group Monitor unique au sein du groupe afin d'éviter des incohérences de décisions à l'issue de l'algorithme de recouvrement. Au terme du vote (procédure basée sur le principe de priorité croissante), le Group Monitor vainqueur suspend le trafic dans le système et ne le rétablit que lorsqu'il a terminé son algorithme de recouvrement.

Cet algorithme est composé de deux phases et se déroule comme suit :

- dans la première phase le "Group Monitor" interroge les processeurs pour qu'ils lui fournissent l'état du dernier message reçu provenant du processeur défaillant.
- dans la seconde phase, le "Group Monitor" diffuse la nouvelle vue du groupe et les décisions qu'il a prises concernant les messages pendants : si le "Group Monitor" est invoqué suite à la défaillance d'un émetteur, il annule le message pendant de cet émetteur, si aucun processeur n'a reçu la validation.

4.3.2.1.2.5. Défaillance du Group Monitor

Lorsque le processeur supportant le Group Monitor défaille, il faut absolument éviter que la suspension du trafic n'entraîne un blocage définitif du système. Pour résoudre ce problème, chaque processeur possède un réveil dont le déclenchement indique la défaillance du Group Monitor.

Dès qu'un processeur détecte la défaillance du Group Monitor, il utilise un mécanisme de reprise du trafic, ce qui permet d'effectuer l'élection d'un nouveau Group Monitor et une réinitialisation de la procédure de recouvrement.

4.3.2.1.3. A propos des cinq propriétés

4.3.2.1.3.1. Diffusion spontanée

Un processeur peut diffuser un message à tout instant, dès lors que son message précédent a été validé/annulé. Ce protocole gère les diffusions spontanées.

4.3.2.1.3.2. Diffusions concurrentes

Ce protocole permet des diffusions concurrentes à l'intérieur d'un groupe. La concurrence des diffusions est gérée directement lors de la phase de dissémination du protocole : en effet, seule la dernière transmission étant prise en compte par l'émetteur, tous les processeurs reçoivent en même temps le même message, ce qui permet donc un ordonnancement total chez tous les processeurs corrects du groupe de diffusion.

4.3.2.1.3.3. Nombre minimal d'annulation de messages

Deux facteurs sont responsables de l'annulation d'un message. Le premier concerne le comportement d'un récepteur incapable de recevoir un message par manque de place (buffers pleins) et le second fait suite à la défaillance d'un émetteur. Dans le premier cas, l'émetteur sur réception d'un rejet diffuse l'annulation de son message dans sa phase de décision. Dans le second cas, le Group Monitor exclut l'émetteur défaillant et annule son message pendant, si aucun processeur n'a reçu la validation.

Remarquons cependant que l'émetteur a pu valider le message avant de défaillir et si aucun processeur n'a reçu la validation, le message est annulé. L'historique de l'émetteur défaillant n'est pas un préfixe de l'historique des processeurs corrects.

4.3.2.1.3.4. Pas d'exclusion abusive

Si le dépassement du degré d'omission n'est pas vérifié en cours de fonctionnement, des exclusions abusives sont tout-à-fait envisageables. En effet, lorsque le Group Monitor est invoqué, il exclut un membre du groupe de diffusion. Donc, si le Group Monitor est appelé par un membre qui soit a dépassé son degré d'omission, soit pour lequel le médium a dépassé son degré d'omission, il peut exclure une majorité de processeurs.

4.3.2.1.3.5. Non contamination du système

La contamination du système peut être due d'une part à la défaillance d'un émetteur et d'autre part, au dépassement du degré d'omission. Considérons le cas oú l'émetteur d'une diffusion défaille en cours de phase de décision ; le Group Monitor, chargé de terminer la diffusion pendante, ne communique pas avec les processeurs ayant reçu la validation de l'émetteur défaillant. Conformément au protocole, le Group Monitor annule le message pendant. Ainsi, certains processeurs auront validé le message, et d'autres l'auront annulé.

Dans AMp, un émetteur qui au terme de sa phase de dissémination ne peut obtenir tous les acquittements exclut les silencieux (les processeurs dont il n'a pas reçu les acquittements relatifs à sa dernière transmission). Le dépassement du degré d'omission du bus peut aboutir à l'existence de plusieurs groupes évoluant concurrement, utilisant des vues différentes et donc finalement aboutir à une incohérence du système.

4.3.2.1.4. Conclusion

AMp est un protocole de diffusion atomique relativement simple et efficace : c'est un protocole à contrôle décentralisé, tolérant les fautes d'omissions, n'annulant un message que si aucun processeur n'a reçu sa validation, mais toutefois, l'algorithme déroulé par le Group Monitor peut poser problème, car il est possible que l'historique de l'émetteur défaillant ne soit pas le préfixe de l'historique des processeurs corrects.

4.4. Récapitulatif des protocoles de diffusion atomique à contrôle décentralisé

PROTOCOLE DE DIFFUSION ATOMIQUE	CHANG - MAXEMCHUCK	GLIGOR - LUAN
Validation	symétrique	asymétrique
Nombre de phases	*(N+L)* passages de jeton avec N nombre de processeur de l'anneau et *L* = cste du protocole	3 phases
mode de défaillance	arrêt sur défaillance	arrêt sur défaillance
Horloge	locale non synchronisée	locale non synchronisée
Degré de parrallélisme	<= nombre de membres	<= *k avec k* = cste du protocole
partitionnement	non toléré	partitionnement toléré si 2 hyp. supplémentaires
algorithme de recouvrement	protocole de reconfiguration de l'anneau : 2 phases	protocole de terminaison invoqué jusqu'à ce que les messages pendants soient décidés par une majorité
Les 5 propriétés		
Diffusion spontanée	oui	oui
Diffusion concurrente	oui	oui
Nombre min. d'annulations	annulation d'un message reçu par moins de *L* processeurs	un message est validé tant qu'il existe une partition majoritaire
Non contamination	contamination possible si L n'est pas au moins égal à la majorité	pas de contamination possible
Exclusion abusive	exclusion abusive possible : 1 seul avis est suffisant pour exclure un membre du groupe	il n'existe pas de protocole d'exclusion

figure 14. Diffusion atomique - contrôle décentralisé - arrêt sur défaillance

PROTOCOLE DE DIFFUSION ATOMIQUE	BIRMAN - JOSEPH *ABCAST*	BIRMAN - SCHIPER STEPHENSON *Fast ABCAST*
Validation	asymétrique	
Nombre de phases	2 phases	2 phases
mode de défaillance	arrêt sur défaillance	arrêt sur défaillance : 1 seule défaillance par groupe de diffusion
Horloge	locale non synchronisée	locale non synchronisée
Degré de parallélisme	<= *k.n* avec *k* = cste du protocole et *n* = nbre. de membres	<= *n* avec *n* = nbre de mbres
partitionnement	non toléré	non toléré
algorithme de recouvrement	algorithme de gestion de vue, exécuté en 2 phases	protocoles indépendants : - monitoring protocol - protocole de gestion de vue
Les 5 propriétés		
Diffusion spontanée	oui	oui
Diffusion concurrente	oui	oui
Nombre min. d'annulations	oui	annulation d'un msg : - émetteur = processeur jeton - défaillance du processeur jeton en cours de diffusion
Non contamination	1) défaillance simultanées de l'émetteur et d'un récepteur 2) 1 émetteur ne communique pas avec une majorité	contamination si : - monitoring protocol - protocole de gestion de vue identiques à [BJ87]
Exclusion abusive	oui	exclusions abusives si : - monitoring protocol - protocole de gestion de vue identiques à [BJ87]

figure 15. protocole de diffusion atomique - contrôle décentralisé - arrêt sur défaillance

PROTOCOLE DE DIFFUSION ATOMIQUE	VERISSIMO - RODRIGUES - BAPTISTA *AMp*
validation	asymétrique
Nombre de phases	2 phases
mode de défaillance	omission + degré d'omission
Horloge	locale
Degré de parallélisme	<= nombre de membres
partitionnement	non toléré
algorithme de recouvrement	protocole du group monitor exécuté en 2 phases
Les 5 propriétés	
Diffusion spontanée	oui
Diffusion concurrente	oui
Nombre min. d'annulations	1) manque de tampon 2) défaillance de l'émetteur
Non contamination	1) contamination sur dépassement du degré d'omission du bus 2) défaillance d'un émetteur : validation par émetteur et annulation par les autres
Exclusion abusive	exclusion abusive possible : le group monitor exclut sur un seul avis

figure 16. Diffusion atomique - contrôle décentralisé - omission

PROTOCOLE DE DIFFUSION ATOMIQUE	CRISTIAN- AGHILI- STRONG- DOLEV
Validation	symétrique
Nombre de phases	1 phase
mode de défaillance	processeur : arrêt adaptateur : omission
Horloge	locale synchronisée
Degré de parallélisme	<= k.n avec k = cste du protocole et n = nbre de membres
partitionnement	non toléré
algorithme de recouvrement	
Les 5 propriétés	
Diffusion spontanée	oui
Diffusion concurrente	oui
Nombre min. d'annulations	oui
Non contamination	violation des bornes : 1) non contrôle du dépassement de f 2) écart entre 2 horloge > ε
Exclusion abusive	non : pas de procédure d'exclusion de processeur

figure 17. Diffusion atomique - contrôle décentralisé- arrêt

Bibliographie

[BCJ90] K. Birman, R. Cooper,T. Joseph, K. Marzullo, H. Makpongou, F. Schmuck, M. Wood, "ISIS System Manual Version 2.0", Mars 1990.

[BJ87] K. Birman, T. Joseph, "Reliable communication in the presence of failures", ACM Transaction on Computer Systems, Vol 5-1, pp.47-76, Août 1987.

[BSS90] K. Birman, A. Schiper, P. Stephenson, "Fast Causal Multicast", Cornell Computer Science Technical Report TR-1105, Avril 1990.

[CAS85] F. Cristian, H. Aghili, R. Strong, D. Dolev, "Atomic broadcast : from simple message diffusion to Byzantine agreement", Proceedings FTCS 15, ann Arbor, 30 pp, 1985.

[CDD90] F.Cristian, B. Dancey, J. Dehn, "Fault-tolerance in the Advanced Automation System", FTCS 20th, Newcastle, UK, pp.6-20, Juin 1990.

[CM84] J. Chang, N. Maxemchuck, "Reliable Broadcast protocols", ACM Transaction on Computer Systems, Vol.2, No.3, August 1984, pp.251-273.

[FLP85] M. Fischer, N. Lynch, M. Paterson, "Impossibility of distributed consensus with one faulty processor", Jal of ACM, Vol.32, No.2, 1985.

[GL90] V. Gligor, W. Luan "A fault-tolerant protocol for atomic broadcast", IEEE Transactions on parallel and distributed systems, Vol. 1, No.3, pp.271-285, July 1990.

[GT89] A. Gopal, S. Toueg, "Reliable Broadcast in Synchronous and Asynchronous environment", 3th International Workshop on Distributed Algorithms, Nice, Septembre 1989.

[GT91] A. Gopal, S. Toueg, "Inconsistency and contamination", 1991 PODC, Montréal, Canada, PP257-272, August 1991.

[GRA78] J. Gray, "Notes on database operating systems" in "Operating systems : an advanced course", Lecture Notes in Computer Science, Springer,Verlag pub, No.60, pp.393-481,1978.

[HAD90] V.Hadzilacos, "On the relationship between the atomic commitment and consensus problems", Lecture Notes in Computer Science, Springer,Verlag pub, Vol.448, pp.201-207, 1990.

[KTH89] F. Kaashoek, S. Tanenbaum, F. Hummel, E. Bal, "An efficient reliable broadcast protocol", ACM Operating Systems Review, Vol. 23, No.4, , pp.5-19, Octobre 1989.

[LAM78] L. Lamport, "Time clocks and the ordering of events in a distributed system", Communication ACM, Vol.21, N7, pp558-565, 1978.

[LSP82] L. Lamport, R. Shostak, M. Pease, "The Byzantine generals problem", ACM Trans. on progamming languages and systems, Vol.4, pp. 382-401, 1982.

[MA91] P. Minet, E. Anceaume, "ABP : an atomic broadcast protocol", Rapport de Recherche No.1473, INRIA, Juin 1991.

[POS80] J. Postel, "Transmission Control Protocol", RFC 761, USC/Information Sciences Institute, Janvier 1980.

[RC91] A. Ricciardi, K. Birman, "Using process groups to implement failure detection in asynchronous environments" ACM Operating Systems Review, Vol. 25, N2, pp341-351, April 1991.

[VRB89] P. Verissimo, P. Rodrigues, M. Baptista, "AMp : A highly parallel atomic multicast protocol", Computer Communication Review, Vol. 19, number 4, pp.83-93, Septembre 1989.

ISSN 0249-6399